

**THE MULTIRATE INTEGRATION PROPERTIES OF
WAVEFORM RELAXATION, WITH APPLICATIONS TO
CIRCUIT SIMULATION AND PARALLEL COMPUTATION**

by

Jacob K. White

Memorandum No. UCB/ERL M85/90

1818 November 1985

Report Documentation Page			Form Approved OMB No. 0704-0188		
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE NOV 1985		2. REPORT TYPE		3. DATES COVERED 00-00-1985 to 00-00-1985	
4. TITLE AND SUBTITLE Multirate Integration Properties of Waveform Relaxation with Application to Circuit Simulation and Parallel Computation			5a. CONTRACT NUMBER		
			5b. GRANT NUMBER		
			5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S)			5d. PROJECT NUMBER		
			5e. TASK NUMBER		
			5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of California at Berkeley, Department of Electrical Engineering and Computer Sciences, Berkeley, CA, 94720			8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSOR/MONITOR'S ACRONYM(S)		
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT Same as Report (SAR)	18. NUMBER OF PAGES 174	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

THE MULTIRATE INTEGRATION PROPERTIES OF WAVEFORM
RELAXATION, WITH APPLICATIONS TO CIRCUIT
SIMULATION AND PARALLEL COMPUTATION

by

Jacob K. White

Memorandum No. UCB/ERL 85/90

18 November 1985

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

**THE MULTIRATE INTEGRATION PROPERTIES OF WAVEFORM RELAXATION, WITH
APPLICATIONS TO CIRCUIT SIMULATION AND PARALLEL COMPUTATION**


Jacob K. White

Ph.D.

Department of Electrical Engineering
and Computer Science

Sponsors: DARPA, GTE, I.B.M.,
T.I., Philips

Signature



Alberto Sangiovanni-Vincentelli
Committee Chairman

ABSTRACT

Because of the high cost of fabricating an Integrated Circuit(IC), it is important to verify the design using simulation. There are a wide variety of techniques for simulating integrated circuit designs, but the most accurate and reliable is to construct the system of nonlinear ordinary differential equations that describe a given circuit, and solve the system with a numerical integration method. This approach, referred to as circuit simulation, is computationally expensive, particularly when applied to large circuits. To reduce the computation time required to simulate large MOS circuits, new numerical integration algorithms based on relaxation techniques have been developed. These techniques can reduce the simulation time as much as an order of magnitude over standard circuit simulation programs. In addition, they are particularly suited for parallel implementation. This thesis covers both the classical numerical techniques and the new relaxation-based algorithms, with particular emphasis on the Waveform Relaxation (WR) family of algorithms. Algorithms in this family are reviewed, convergence theorems are included, and their implementations on a parallel processor are presented.

ACKNOWLEDGEMENTS

I would like to thank my research advisor, Professor Alberto L. Sangiovanni-Vincentelli, for introducing me to the area of circuit simulation, and for his professional guidance and personal inspiration. His belief about CAD research, that one should work just as hard on proving the theorem as on getting good practical results, is one that I will carry with me always. Also, I would like to thank him for not really meaning it when he said "Next time, Mr White, I am going to kick you out of the CAD group."

I appreciate the fellowship from GTE, which has given me financial support in pursuing my graduate study at U.C. Berkeley. I also appreciate the research grants from DARPA and the IBM corporation.

I wish to express my thanks to Farouk Odeh at IBM's T. J. Watson Research Center, for the discussions from which many of the theorem proofs in this thesis were derived, and for being my friend in spite of the Israel-Palestine situation. In addition, I would like to thank J. Beetem, W. Donath, and P. Defebve at IBM's T. J. Watson Research Center for many valuable discussions and suggestions.

Several Berkeley professors have graciously contributed their time on my behalf, and I would like to express my gratitude. Prof. W. Kahan provided many insights into the nature of numerical integration methods, Prof. K. Miller reviewed this thesis, Prof. D. O. Pederson has been a source of continued support, Prof. A. R. Newton reviewed this thesis, and through discussion offered many suggestions on circuit simulation techniques. Along with Prof. Desoer and Prof. A. Sangiovanni-Vincentelli these professors agreed to serve on my qualifying committee, and I would especially like to thank them for making my qualifying exam an experience I will not soon forget.

I would like to thank Palab Chatterjee, Paul Cox, and Ping Yang at Texas Instruments, S. Greenberg and K. Sakallah at Digital Equipment Corporation, and C-F. Chen, L. Nagel, and P.

Subrahmanyam at AT&T Bell Laboratories, both for providing state-of-the-art circuit examples, and for their insight into industrial use of simulation tools.

Several people participated in the development of the RELAX2.3 program and I would like to acknowledge their contributions and express my gratitude for their help. Peter Moore wrote the input processor, Ken Kundert wrote the sparse matrix package, Resve Saleh wrote the MOS level 3 model evaluation code, and Tom Quarles suggested many of the programming techniques.

Much of my research could not have been completed without Diet Coke, a fine product from the Coca-Cola company. I would like to thank Tom Quarles and Peter Moore for insuring that a steady supply of Diet Coke was always available.

In the several years that I have been a member of the Berkeley CAD group, I have had the privilege of exchanging ideas with almost every other member. I have enjoyed the many discussions I've had with Ken Kundert and Rick Rudell and I would like to thank them for assuming the important task of Sangiovanni-baiting. I have enjoyed the many collaborations with Resve Saleh, and would like to thank him for an uncountable number of favors, both large and small. It has also been my pleasure to exchange ideas about circuit simulation with G. Casinovi, G. DeMicheli, R. Gyurcsik, J. Kleckner, G. Marong, K. Mayaram, D. Ryan, R. Rudell, T. Huang and D. Webber.

Finally, I would like to express my appreciation to my parents Melvin and Anne White, and to my wife, Barbara Bratzel.

CONTENTS

CHAPTER 1 - INTRODUCTION	1
CHAPTER 2 - THE CIRCUIT SIMULATION PROBLEM	6
SECTION 2.1 - THE EQUATION SYSTEM	6
SECTION 2.1.1 - CONSTRUCTING THE EQUATION SYSTEM	7
SECTION 2.1.2 - EXTENDING THE CONSTRUCTION TECHNIQUE	10
SECTION 2.2 - NUMERICAL INTEGRATION PROPERTIES	12
SECTION 2.2.1 - CONSISTENCY, STABILITY, AND CONVERGENCE	13
SECTION 2.2.2 - STIFFNES AND A-STABILITY	17
SECTION 2.2.3 - CHARGE CONSERVATION	19
SECTION 2.2.4 - DOMAIN OF DEPENDENCE	23
CHAPTER 3 - NUMERICAL TECHNIQUES	33
SECTION 3.1 - NUMERICAL INTEGRATION IN GENERAL-PURPOSE SIMULATORS ..	33
SECTION 3.2 - RELAXATION DECOMPOSITION	39
SECTION 3.3 - SEMI-IMPLICIT NUMERICAL INTEGRATION METHODS	45
SECTION 3.4 - RELAXATION VS SEMI-IMPLICIT INTEGRATION	50
CHAPTER 4 - THE WAVEFORM RELAXATION ALGORITHM	52
SECTION 4.1 - THE BASIC WR ALGORITHM	53
SECTION 4.2 - CONVERGENCE PROOF FOR THE BASIC WR ALGORITHM	55
SECTION 4.3 - NONSTATIONARY WR ALGORITHMS	62
SECTION 4.4 - WAVEFORM RELAXATION-NEWTON METHODS	65

CHAPTER 5 - DISCRETIZED WR ALGORITHMS	70
SECTION 5.1 - THE GLOBAL TIMESTEP CASE	71
SECTION 5.2 - GLOBAL FIXED-TIMESTEP WR CONVERGENCE THEOREM	76
SECTION 5.3 - THE MULTI-RATE WR CONVERGENCE THEOREM	90
 CHAPTER 6 - ACCELERATING WR CONVERGENCE	 100
SECTION 6.1 - UNIFORMITY OF WR CONVERGENCE	100
SECTION 6.2 - PARTITIONING LARGE SYSTEMS	110
 CHAPTER 7 - THE IMPLEMENTATION OF WR IN RELAX2.3	 127
SECTION 7.1 - PARTITIONING MOS CIRCUITS	128
SECTION 7.2 - ORDERING THE SUBSYSTEM COMPUTATION	131
SECTION 7.3 - COMPUTATION OF THE SUBSYSTEM WAVEFORMS	134
SECTION 7.4 - WINDOWSIZE DETERMINATION	137
SECTION 7.5 - PARTIAL WAVEFORM CONVERGENCE	139
SECTION 7.6 - EXPERIMENTAL RESULTS	140
 CHAPTER 8 - PARALLEL WR ALGORITHMS	 143
SECTION 8.1 - A BRIEF OVERVIEW OF THE SHARED MEMORY COMPUTER	143
SECTION 8.2 - A MIXED GAUSS-SEIDEL/JACOBI PARALLEL WR ALGORITHM	147
SECTION 8.3 - TIMEPOINT-PIPELINING WR ALGORITHM	150
SECTION 8.4 - PARALLEL ALGORITHM TEST RESULTS	153
 CHAPTER 9 - CONCLUSIONS	 155
 REFERENCES	 157

CHAPTER 1 - INTRODUCTION

Reliable and accurate simulation tools must play a key role in Integrated Circuit (IC) design. This is because fabricating an integrated circuit is expensive and often time-consuming (on the order of months). In addition, minor errors in the integrated circuit design can not usually be corrected after fabrication. Therefore, design errors must be uncovered before fabrication, and this can be done through the use of simulation.

There are a wide variety of techniques for simulating integrated circuit designs, but none are as accurate, reliable, and technology independent as constructing the system of nonlinear ordinary differential equations that describe a given circuit, and solving this system with a numerical integration method. This approach, referred to as circuit simulation, has been implemented in a variety of programs such as SPICE[2] or ASTAP[3]. These programs use a standard, or direct, techniques based on the following four steps:

- i) An extended form of the nodal analysis technique to construct a system of the differential equations from the circuit topology.
- ii) Stiffly stable implicit integration methods, such as the Backward Difference formulas, to convert the differential equations which describe the system into a sequence of nonlinear algebraic equations.
- iii) Modified Newton methods to solve the algebraic equations by solving a sequence of linear problems.
- iv) Sparse Gaussian Elimination to solve the systems of linear equations generated by the Newton method.

Circuit simulation tools based on the above techniques are heavily used. Companies spend many millions of dollars per year in computer costs, and a number of companies run over 60,000 simulations/month. However, these programs were designed in the early 1970's for the simulation of circuits with a few hundred transistors at most. They are now being applied, somewhat inappropriately, to the task of simulating digital and analog VLSI circuits, which can contain more than 50,000 devices. As problems increase in size, it becomes less economically feasible to use the above

direct techniques. SPICE[2] and ASTAP[3] can take several hours (on a VAX11/780) to simulate circuits with only a few hundred devices.

There are two reasons why the direct approach described above can become inefficient for large systems. The most obvious reason is that sparse matrix solution time will grow super-linearly with the size of the problem. Experimental evidence indicates that the point where the matrix solution time begins to dominate is when the system has over several thousand nodes, and this is the size of systems that are beginning to be simulated for new IC designs.

The direct methods become inefficient for large problems also because, for large differential equation systems, the different state variables are changing at very different rates. Direct application of the integration method forces every differential equation in the system to be discretized identically, and this discretization must be fine enough so that the fastest changing state variable in the system is accurately represented. If it were possible to pick different discretization points, or time-steps, for each differential equation in the system so that each could use the largest time-step that would accurately reflect the behavior of its associated state variable, then the efficiency of the simulation would be greatly improved.

Several modifications of the direct method have been used that both avoid large sparse matrix solutions, and allow the individual equations of the system to use different time-steps [4,5,6,7,8,9,10,11]. One class of such techniques, Waveform Relaxation[11,12,13,14,15,16,17,18] is based on "lifting" the Gauss-Seidel and Gauss-Jacobi relaxation techniques for solving large algebraic systems to the problem of solving the large systems of ordinary differential equations associated with MOS digital circuits. Briefly, the idea of these relaxation technique is to first break a large circuit into loosely coupled subcircuits. Then the behavior of each subcircuit, over some interval of time, is calculated by "guessing" the behavior of the surrounding subcircuits over the same interval of time. The responses for each subcircuit are used to improve these guesses, and the response is recalculated. The procedure is iterated until the convergence is achieved for each subcircuit over the interval of time. Other relaxation techniques such as the Gauss-Seidel-Newton algorithm [21] can be applied to

solve the nonlinear system of algebraic equations in place of the standard Newton-Raphson techniques.

Two circuit simulation programs have been developed at Berkeley using relaxation techniques: RELAX, based on Waveform Relaxation[11,18] and SPLICE, based on Iterated Timing Analysis (ITA) [33], a form of Gauss-Seidel-Newton technique. On a uniprocessor, these programs can show speed improvements over direct methods of up to an order of magnitude even for problems with only a few hundred devices. In addition, both the ITA and Waveform Relaxation are particularly amenable to use on multiprocessors because the computational method already decomposes the problem. A distributed form of the ITA algorithm, called DITA, has been recently developed and a prototype DITA simulator, the MSPLICE program, has been implemented[34].

In this thesis I present a complete and consistent study of the existing body of research relating to the application of numerical integration methods differential systems that describe circuits. I then present new theoretical and practical results on the application of WR to numerically solving the differential equations generated from circuits, both on serial and parallel processors.

I start in Chapter 2 with an introduction to the circuit simulation problem, beginning with how the differential equations that describe a circuit are formulated from the circuit topology. Then, those aspects of the circuit simulation problem that play a role in the choice of numerical method are described. The well-known issues of consistency and stiff stability[1] is mentioned briefly, as is a consistent interpretation of the *charge conservation* property[41]. The chapter is ended with the description of a new property that can be used to classify integration methods, that of *exhaustive domain of dependence*.

In Chapter 3, many of integration methods that have been applied to circuit simulation problems are analyzed with respect to the properties described in Chapter 2. The standard multistep integration methods are analyzed first, and it is proved that the implicit multistep integration methods commonly used in circuit simulation have all the desirable properties given in Chapter 2. Following, the relaxation algorithms that have been used to solve the large algebraic systems generated by im-

implicit integration methods is described[21,33], and a theorem guaranteeing the convergence of such methods for small timesteps is proved[12]. Then, the semi-implicit integration algorithms, used in special purpose timing simulation programs[5,6,7,8], are analyzed with respect to their domain of dependence and stability properties. The chapter is ended by comparing the semi-implicit and relaxation algorithms.

The theoretical basis for the family of WR algorithms, methods for the decomposed solution of differential equations, is presented in Chapter 4. Waveform relaxation is introduced with a simple example followed by a general algorithm. Then a new proof of the WR convergence, one that demonstrates that the WR algorithm is a contraction mapping in a particular norm, is presented. Extensions to the basic algorithm that allow for modified iteration equations is presented and it is shown that the convergence of such extensions follows *directly* from the proof that the WR algorithm is a contraction mapping. Following, an extension of the Newton Method to function spaces is presented, and its convergence proved using lemmas from the basic theorem. The waveform Newton algorithm will then be combined with the WR algorithm to produce a waveform relaxation-Newton(WRN) algorithm[22].

To compute the iteration waveforms for the WR algorithm it is usually necessary to solve systems of nonlinear ordinary differential equations. If multistep integration formulas are used to solve for the iteration waveforms, the numerical integration method plays a role in the convergence properties of this discretized WR algorithm[29]. In Chapter 5, the interaction between WR algorithms and multistep integration methods is considered in detail. The discretized WR algorithm will be analyzed first assuming that every differential equation in the system is discretized identically (the global-timestep case). A simple example is presented that demonstrates a possible breakdown of the WR method under discretizations. Then, a comparison is drawn between the discretized WR algorithm and the algebraic relaxation methods described in Chapter 3 and a strong comparison theorem for linear systems is proved. Following, a convergence theorem for the fixed global-timestep

discretized WR algorithm will then be presented. The global-timestep restriction will then be lifted, and the first theorem proving the convergence of the multi-rate WR relaxation algorithm is presented.

In Chapter 6, the theoretical background for two of the techniques for accelerating WR convergence is presented. First, why breaking the simulation interval into pieces, called windows, can be used to reduce the number of relaxation iterations required to achieve convergence is examined[17], and then how to partition large systems into subsystems in such a way that the WR algorithm converges rapidly is considered[31].

The implementation of the WR algorithm in the RELAX2.3 program is described in Chapter 7. The partitioning, numerical integration, windowing and partial waveform convergence algorithms as applied to MOS circuits are presented. The results from simulating a CMOS memory circuit are analyzed, in order to demonstrate more clearly both the practicality of the WR algorithm, and the specific nature of its efficiencies. The chapter will be concluded with a table of results from the RELAX2.3 program applied to a variety of MOS circuits.

The implementation of two WR-based parallel circuit simulation algorithms on a shared-memory computer are described in Chapter 8[17]. A brief overview of the aspects of a shared-memory computer that effect the algorithm implementation are presented, followed by the description of, and experimental results from, the two parallel WR algorithms.

CHAPTER 2 - THE CIRCUIT SIMULATION PROBLEM

As mentioned in the introduction, circuit simulation amounts to solving numerically the system of nonlinear ODE's that describe the dynamic behavior of a circuit. In this Chapter, we will address the two topics of the construction of a system of differential equations from a given circuit topology and its properties, and the issues to consider when choosing a numerical method for solving that system.

SECTION 2.1 - THE EQUATION SYSTEM

The most general formulation of a system of nonlinear differential equations is the following implicit formulation:

$$F(\dot{x}(t), x(t), u(t)) = 0 \quad x(0) = x_0 \quad [2.1]$$

where $x(t) \in \mathbb{R}^n$ on $t \in [0, T]$; $u(t) \in \mathbb{R}^r$ on $t \in [0, T]$ is piecewise continuous; and $F: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^r \rightarrow \mathbb{R}^n$ is continuous.

Before considering techniques for numerical solution, we first must guarantee that Eqn. (2.1) has a solution. If we require that there exists a transformation of Eqn. (2.1) to the form $\dot{y} = f(y, u)$ where f is Lipschitz continuous with respect to y for all u , then a unique solution for the system exists[39]. Although there are many sets of broad constraints on F that guarantee the existence of such a transformation, the conditions can be difficult to verify in practice. Rather than carefully considering the existence question, which will complicate the analyses that follow without lending much insight, we will consider the following less general form, in which most circuit simulation problems can be described.

$$C(x(t), u(t)) \dot{x}(t) = f(x(t), u(t)) \quad x(0) = x_0 \quad [2.2]$$

where $x(t) \in \mathbb{R}^n$ on $t \in [0, T]$; $u(t) \in \mathbb{R}^r$ on $t \in [0, T]$ is piecewise continuous; $C: \mathbb{R}^n \times \mathbb{R}^r \rightarrow \mathbb{R}^{n \times n}$ is such that $C(x, u)^{-1}$ exists and is uniformly bounded with respect to x, u ; and $f: \mathbb{R}^n \times \mathbb{R}^r \rightarrow \mathbb{R}^n$ is globally Lipschitz continuous with respect to x for all $u(t) \in \mathbb{R}^r$.

The fact that $C(x, u)$ has a well-behaved inverse guarantees the existence of a normal form for Eqn. (2.2), and that $x(t) \in \mathbb{R}^n$ is the vector of state variables for the system. Then as f is globally Lipschitz continuous with respect to x for all u , $C(x, u)^{-1}$ is uniformly bounded, and $u(t)$ is piecewise continuous, there exists a unique solution to Eqn. (2.2) on any finite interval $[0, T]$ [39].

SECTION 2.1.1 - CONSTRUCTING THE EQUATION SYSTEM

The behavior of the most commonly modeled nonlinear circuit elements: diodes, bipolar transistors, and MOS transistors, can be described by *voltage-controlled* current and charge equations. For example, consider the diode in Fig. 2.1 for the case where the voltage across the diode $v_{ac} < 0.0$. Then the anode and cathode currents, i_a and i_c respectively, and the anode and cathode charges, q_a and q_c respectively, can be computed (to first order) from the following equations,

$$i_a = I_s(e^{v_{ac}/V_t} - 1)$$

$$i_c = -i_a$$

$$q_a = C_0 \left(\frac{1 - v_{ac}}{\phi} \right)^{1-m}$$

$$q_c = -q_a$$

where I_s is the saturation current, V_t is the thermal voltage, C_0 is the zero-bias junction capacitance, and ϕ is the junction potential.

For an arbitrary circuit made up of a network of elements described by voltage-controlled current and charge equations, it is possible to construct a system of differential equations that describes the circuit by using nodal analysis[36]. This amounts to applying the relationship that the

time derivative of charge, q , is a current, and insisting that the sum of the currents leaving each node (currents entering the node are assigned negative sign) in the network is precisely zero (Kirkchoff's Current Law, KCL). That is, for each node in the network:

$$\frac{d}{dt} \sum_{\text{elements at } i} q_{\text{element}}(v(t), u(t)) + \sum_{\text{elements at } i} i_{\text{element}}(v(t), u(t)) = 0. \quad [2.3]$$

where $v(t) \in \mathbb{R}^n$ is the vector of node voltages, and $u(t) \in \mathbb{R}^r$. If a system were constructed using the KCL equations for every node in the circuit, the system would be overdetermined. For this reason, the equation for an arbitrary node in the circuit, referred to as the *reference* or *ground* node, is discarded. In addition, the KCL equations for the nodes for which the node voltage is known *a priori* (e.g., a node connected to a voltage source whose other terminal is connected to the reference node) are discarded.

As an example, consider the *Nand* circuit in Fig. 2.2. In order to solve for the unknown voltages v_1 and v_2 , we need only form the KCL equations at node 1 and node 2, and can ignore the KCL equations for the nodes connected to the voltage source and ground. For the first node we have the equation:

$$i_{d_{m1}}(v_1, V_a, 0) - i_{d_{m2}}(v_2, V_b, v_1) + \frac{d}{dt} [q_{d_{m1}}(v_1, V_a, 0) + q_{s_{m2}}(v_2, V_b, v_1) + c_1 v_1] = 0$$

and for the second node,

$$i_{d_{m2}}(v_2, V_b, v_1) + g_1(V_{dd} - v_2) + \frac{d}{dt} [q_{s_{m2}}(v_2, V_b, v_1) + c_2 v_2] = 0$$

where $i_{d_{m1}}$ and $i_{d_{m2}}$ are the the currents flowing from the drain to the source of transistor $m1$ and $m2$ respectively, $q_{d_{m1}}$, $q_{d_{m2}}$, $q_{s_{m2}}$, are the charges accumulated at the drain of transistor $m1$ and the source and drain of transistor $m2$ respectively.

In general, the nodal analysis leads to systems of the form:

$$\frac{d}{dt}q(v(t), u(t)) - i(v(t), u(t)) = 0 \quad [2.4]$$

where $q: \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ is the vector of the sums of the charges at a node, $i: \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ is the vector of the sums of the currents entering a node, $v \in \mathbb{R}^n$ is the node voltage vector, and $u \in \mathbb{R}^m$ is the vector of inputs. The system in Eqn. (2.4) can be converted to the form of Eqn. (2.2) by applying the chain rule to establish the identity

$$\frac{\partial}{\partial t}q(v(t), u(t)) = \frac{\partial q}{\partial v}(v(t), u(t))\dot{v}(t) + \frac{\partial q}{\partial u}(v(t), u(t))\dot{u}(t).$$

We then define $x = v$, $C(x(t), u(t)) = \frac{\partial q}{\partial v}(v(t), u(t))$, and $f(x(t), u(t)) = i(v(t), u(t)) - \frac{\partial q}{\partial u}(v(t), u(t))\dot{u}(t)$ to get a system of the form of Eqn. (2.2). Note that in order for the f defined above to satisfy the Lipschitz continuity property, either $\frac{\partial q}{\partial u}$ must be zero, or \dot{u} must be bounded.

For a broad class of circuits, the $C(x, u)$ matrix defined by $C(x, u) = \frac{\partial q}{\partial v}(v, u)$ is strictly diagonally dominant uniformly in x , a property which guarantees the existence of a bounded inverse[28]. Many of results concerning relaxation methods for systems of the form of Eqn. (2.2) rely on this diagonal dominance property, so we will describe under what conditions a circuit will produce a $C(x, u)$ that is diagonally dominant.

Consider the two node example in Fig. 2.3. Applying the nodal analysis technique described above yields the following differential equations:

$$(c_1 + c_f)\dot{v}_1(t) - c_f\dot{v}_2(t) = g_1v_1(t)$$

$$(c_2 + c_f)\dot{v}_2(t) - c_f\dot{v}_1(t) = g_2v_2(t)$$

As this example demonstrates, for circuits whose only charge elements are capacitors, the i^{th} diagonal entry of the C matrix is the sum of the capacitance incident at node i , and the ij^{th} entry is the negative value of the capacitance between node i and node j . It therefore follows that the sum of the absolute value off-diagonal terms is less than or equal to the diagonal terms where strict inequality holds if there a nonzero capacitance between node i and a voltage source or ground node. This example leads to the following important observation which is easily verified.

Observation If a system of equations of the form of Eqn. (2.2) is constructed by applying the nodal analysis technique described above to a circuit which contains capacitors (linear or nonlinear), or any other elements whose charge function has a diagonally dominant Jacobian, then the capacitance matrix $C(x,u)$ of Eqn. (2.2) is diagonally dominant. If, in addition, there exist a linear or nonlinear capacitor, bounded away from zero, to ground or a voltage source at each node in the circuit, the matrix $C(x,u)$ is strictly diagonally dominant for all x, u .

SECTION 2.1.2 - EXTENDING THE CONSTRUCTION TECHNIQUE

The nodal analysis technique can only be used to form the differential equations of circuits with elements whose current or charge is a well-behaved function of voltage. It is possible to extend the technique to include circuits with inductors and floating voltage sources by using Modified Nodal Analysis [38]. A similar technique is used in this section to show that circuits with these two types elements can be described by a differential equation system of the form of Eqn. (2.2). This demonstrates that the form of Eqn. (2.2) can encompass much more than just circuits with voltage-controlled current and charge elements, and is a justification for considering only systems of the form of Eqn. (2.2) for rest of this thesis.

Consider a large network with two nodes that are connected by a floating voltage source as in Fig. 2.4. The nodal analysis equations can be written for the two nodes and are for node a ,

$$\sum_{j=1}^k i_j(v_a, v_b, v) + i_{src} = 0$$

for node b ,

$$\sum_{j=k+1}^l i_j(v_a, v_b, v) - i_{src} = 0$$

where v is the vector of all the other node voltages and i_{src} is the current through the voltage source. Given an additional variable has been introduced, i_{src} , an additional equation is needed to compute the solution,

$$v_a = v_b + V.$$

In order to convert this set of equations into the form of Eqn. (2.2) we perform a simple substitution to generate one equation in one unknown (here we have arbitrarily chosen v_b)

$$\sum_{j=1}^k i_j(v_b + V, v_b, v) = 0$$

It is somewhat more complicated to reorganize the equations of circuits with inductors so that they fit into the form of Eqn. (2.2). This is because the voltage across the inductor is a function of the time derivative of current passing through it. For the example in Fig. 2.5a, the KCL equation for node a is

$$\sum_{i=1}^j i_i(v_a, v_b, v) + i_{ind} = 0$$

and for node b ,

$$\sum_{i=j+1}^k i_i(v_a, v_b, v) - i_{ind} = 0$$

and for the inductor,

$$L \frac{di_{ind}}{dt} - (v_a - v_b) = 0$$

where v_a and v_b are the voltages at the inductor terminals; v is the vector of node voltages for the entire circuit excluding v_a and v_b ; i_{ind} is the inductor current, and L is its inductance.

Since the derivative of inductor current is present in the equations, in order to include the inductor in the system of Eqn. (2.2), the current must be included in the set of state variables. A circuit interpretation of such a reorganization is to replace the inductor by an extra circuit node, a grounded capacitor of capacitance L , and two voltage-controlled current sources (See fig. 2.5b). Note that the extra row in Eqn. (2.2) that would be generated by including an inductor in a given circuit will not destroy the invertibility or strict diagonal dominance property of $C(x,u)$, because the extra row in $C(x,u)$ will contain only one nonzero entry, on the diagonal.

SECTION 2.2 - NUMERICAL INTEGRATION PROPERTIES

Once the system of differential equations has been constructed from the circuit topology, it must be solved numerically. The usual approach is to use one of the many numerical integration formulas to convert the differential equations which describe the system into a sequence of nonlinear algebraic equations.

For example, the most obvious numerical integration formula is the *explicit-Euler* algorithm. Given the initial condition $x(0) = x_0$, it is possible to compute an approximation to $x(h)$, $h > 0$, by substituting $\frac{\hat{x}(h) - x(0)}{h}$ for $\dot{x}(0)$, where the notation \hat{x} is used to indicate numerical approxi-

mation. Substituting this discrete approximation into Eqn. (2.2) yields the following equation for $\hat{x}(h)$:

$$\hat{x}(h) = x(0) + C(\hat{x}(0), u(0))^{-1} f(x(0), u(0)) \quad [2.5]$$

By substituting $\hat{x}(h)$ for $x(0)$ in Eqn. (2.5) it is possible to compute $\hat{x}(2h)$, and the process can be repeated to produce a sequence that approximates the exact solution to the differential equation at discrete points in time.

The explicit-Euler algorithm is the simplest of a wide variety of discretization techniques for numerically solving large systems of differential equations. In order to choose a discretization method that will be efficient and accurate for a given class of problems, it is necessary to consider several properties of the integration method with respect to the class. In this section we will consider several of the key aspects of the circuit simulation problem that impact the choice of numerical method. We will start by presenting the general classical consistency/stability/convergence criteria both for completeness, and as a vehicle for presenting the notation that will be used throughout this thesis. We will then consider more specific properties of the circuit simulation problem, starting with the well-known issue of stiffness. Following, the properties of *charge conservation* and *domain of dependence* will be defined, and in each case we will consider the impact these properties have on the choice of numerical method.

SECTION 2.2.1 - CONSISTENCY, STABILITY, AND CONVERGENCE

In general, a numerical integration formula produces a sequence approximation to the solution of a differential equation by repeated application, starting from some initial condition x_0 . We will denote the approximation produced by the m^{th} application of a given numerical integration formula to Eqn. (2.2) by $\hat{x}(\tau_m)$, where $\tau_m \in \mathbb{R}$ is such that $\hat{x}(\tau_m)$ is the numerical approximation to the exact solution at $t = \tau_m$. It will be assumed that if the differential equation is to be solved numerically on $[0, T]$, that there exists some finite integer M , such that $\tau_M = T$. In addition, we will refer to

$h_m = \tau_m - \tau_{m-1}$ as the m^{th} discretization timestep. Finally, we will denote the entire sequence $x(\tau_m)$, $m \in \{0, \dots, M\}$ by $\{x(\tau_m)\}$.

If a numerical integration algorithm is to be of any use, it must be possible to arbitrarily accurately approximate the exact solution to the differential equation system uniformly over $[0, T]$ by reducing the discretization timesteps. An integration method with this property is said to be *convergent*, defined formally as follows:

Definition 2.1: Let the discretization timesteps be fixed; that is $h_m = \frac{T}{M}$ for all $m \in \{0, \dots, M\}$. A Numerical integration method is *convergent* with respect to Eqn. (2.2) if the *global error*, defined by

$$\max_{m \leq M} \|\hat{x}(\tau_m) - x(\tau_m)\| \quad [2.6]$$

goes to zero as $M \rightarrow \infty$ ■.

For a numerical integration method to be convergent, it must have two properties. The error made in one timestep must go to zero rapidly as the timestep decreases, and the errors should not grow too rapidly over the timesteps. The error made in one timestep is called the *local truncation error* (LTE).

Definition 2.2: Let $\hat{x}(\tau_m)$ be generated by applying one step of a numerical integration formula to a system of the form of Eqn. (2.2) given the sequence $\{\hat{x}(\tau_{\tilde{m}})\}$, $\tilde{m} < m$ such that $\hat{x}(\tau_{\tilde{m}}) = x(\tau_{\tilde{m}})$. Then the *local truncation error* is defined as $\|\hat{x}(\tau_m) - x(\tau_m)\|$. ■

The best that one could hope to show for general systems is that the global error for the approximation, that is $\max_{m \leq M} \|\hat{x}(\tau_m) - x(\tau_m)\|$, is a function of the sum of the local truncation errors, $\sum_{m=0}^M LTE_m$, where LTE_m is the local truncation error at the m^{th} timestep. Given a fixed interval $[0, T]$, and that $M = \frac{T}{h}$, this sum is bounded below by $\frac{T}{h} LTE_{\min}$ where LTE_{\min} is the minimum of the LTE's over all m . If this sum is to go to zero as $h \rightarrow 0$, then

$$\lim_{h \rightarrow 0} \frac{LTE_m}{h} \rightarrow 0$$

This property is known as *consistency* [1] and is shared by any "reasonable" numerical integration method.

As an example, it is possible to verify that the explicit-Euler algorithm is *consistent* for systems of the form of Eqn. (2.2), by using a Taylor series expansion about $x(\tau_m)$. That is,

$$x(\tau_{m+1}) = x(\tau_m) + h_{m+1}\dot{x}(\tau_m) + \frac{h_{m+1}^2}{2}\ddot{x}(\tilde{\tau})$$

where $\tilde{\tau} \in [\tau_m, \tau_{m+1}]$. From Eqn. (2.5) we get

$$\hat{x}(\tau_{m+1}) = x(\tau_m) + h_{m+1}C(x(\tau_m), u(\tau_m))^{-1}f(x(\tau_m), u(\tau_m)).$$

Substituting for \dot{x} using the following identity,

$$C(x(\tau_m), u(\tau_m))^{-1}f(x(\tau_m), u(\tau_m)) = \dot{x}(\tau_m)$$

and then subtracting,

$$\hat{x}(\tau_{m+1}) - x(\tau_{m+1}) = \frac{h_{m+1}^2}{2}\ddot{x}(\tilde{\tau}) \quad [2.7]$$

which verifies consistency.

Consistency is not sufficient to guarantee that a numerical integration method is convergent. Consistency only insures that the local errors are small, but does not indicate anything about how the errors propagate from one timestep to the next. To insure convergence we need to verify that the numerical integration method has a second property, that of stability[1].

Definition 2.3: A numerical integration method applied to Eqn. (2.2) is *stable* if there exists an h_0 and a constant $K < \infty$ such that for any two different initial conditions x_0 and x'_0 , and any $h = \frac{T}{N} < h_0$,

$$\|\hat{x}(\tau_M) - \hat{x}'(\tau_M)\| < K \|x_0 - x'_0\|. \blacksquare$$

The explicit-Euler algorithm is *stable*, but the proof is lengthy and well-documented elsewhere[1] so we will not repeat it here.

Not surprisingly, we have the following classical result:

Theorem 2.1: If a numerical integration method is consistent and stable with respect to Eqn. (2.2), then it is convergent with respect to Eqn. (2.2). ■

Several different proofs have been given for this basic result[1].

If an integration method is convergent then when the method is used to compute an approximate solution to a differential equation system, sufficient accuracy can be insured by using timesteps that are small enough. Obviously, it is possible to insure that the timesteps are small enough by using extremely small timesteps, but this is very inefficient. Instead, the integration timesteps are usually controlled by using some check on the discretization error. If, in any given step the error becomes too large, the timestep is replaced by a smaller timestep.

Usually, the check on the discretization error is some computed estimate of the local truncation error. For the explicit-Euler algorithm, for example, the exact local truncation error at the m^{th} step is $0.5h_{m+1}^2\ddot{x}(\tilde{\tau})$ where $\tilde{\tau} \in [\tau_m, \tau_{m+1}]$. An estimate of the local truncation error of the m^{th} explicit-Euler step can be computed using the following divided-difference estimate for \ddot{x} ,

$$\ddot{x}(\tilde{\tau}) \approx \frac{\frac{\hat{x}(\tau_{m+1}) - \hat{x}(\tau_m)}{h_{m+1}} - \frac{\hat{x}(\tau_m) - \hat{x}(\tau_{m-1})}{h_m}}{0.5(h_{m+1} + h_m)}. \quad [2.8]$$

Most of the techniques for estimating local truncation error are only estimates, not bounds. In practice, these type of estimates have proved to be reliable, but there are certain common cases where

the estimates are much smaller than the actual error. An example of such a case will be presented in Section 2.2.4.

SECTION 2.2.2 - STIFFNES AND A-STABILITY

Consider the Example in Fig. 2.6, a resistor-capacitor circuit. The differential equation that describes the circuit can be constructed using the nodal analysis technique above and is:

$$\dot{v}(t) = -100v(t) \quad v(0) = 1.0 \quad [2.9]$$

where $v(t) \in \mathbb{R}$ is the node voltage. The exact solution for v is $v(t) = e^{-100t}$. If the interval of interest is $[0, T]$, this is a two time-scale problem. That is, v changes very rapidly compared to the interval of interest.

Any system of differential equations that has the kind of multiple time-scale properties of the above example is said to be *stiff*. Most circuits of interest generate stiff differential equation systems, and this strongly effects the choice of integration formulas. For example, the explicit-Euler algorithm applied with a fixed timestep h to numerically solve Eqn. (2.9), yields the following recursion equation for \hat{v} ,

$$\hat{v}(\tau_m) = (1 - 100h_m)\hat{v}(\tau_{m-1})$$

or given $v(0) = 1$,

$$\hat{v}(\tau_m) = \prod_{i=1}^m (1 - 100h_i).$$

Clearly, $|\hat{v}(\tau_m)|$ will decay only if $h_m < 0.02$ for all m , and $\hat{v}(\tau_m)$ will decay monotonically to 0 only if $h_m < 0.01$ for all m . If larger timesteps are used, $|\hat{v}(\tau_m)|$ will grow. What this implies is that in

order to accurately compute a sequence approximation to the solution of this system using explicit-Euler, small timesteps must be used *even when the solution is not changing appreciably*.

Now consider a slightly different numerical integration formula, the implicit-Euler algorithm, where $\hat{v}(\tau_m)$ is approximated by $\frac{1}{h_m}(\hat{v}(\tau_m) - \hat{v}(\tau_{m-1}))$. Just like explicit-Euler, implicit-Euler is convergent, and the local truncation error is of order h^2 . When applied to Eqn. (2.9) the following recursion equation results:

$$\hat{v}(\tau_m) = \hat{v}(\tau_{m-1}) - 100 h_m \hat{v}(\tau_m)$$

or reorganizing,

$$\hat{v}(\tau_m) = \frac{1}{(1 + 100h_m)} \hat{v}(\tau_{m-1}).$$

Again using the fact that $v(0) = 1$,

$$\hat{v}(\tau_m) = \prod_{i=1}^m (1 + 100h_i)^{-1}$$

Note that in this case, any $h_m > 0$ will produce a monotonically decaying sequence. The tremendous advantage of this method over explicit-Euler is that small timesteps can be used for the first few steps to accurately resolve the rapid decay, and when the solution stops changing appreciably, the timestep can safely be made orders of magnitude larger without causing the computed solution to grow.

The implicit-Euler algorithm has a property that is "stronger" than the *numerical stability* of Definition 2.3, which we define below as *A-stability*:

Definition 2.4: Let $\{\hat{x}(\tau_m)\}$ be the sequence generated by a numerical integration method applied to the equation

$$\dot{x}(t) = Ax(t) \quad x(0) = x_0$$

where $x(t) \in \mathbb{R}^n$, and $A \in \mathbb{R}^{n \times n}$ and $\tau_m - \tau_{m-1} = h_m = h$ for all m . Given $\{\lambda_i\}$, the set of eigenvalues of A , the *region of stability* for the integration method is the subset of \mathbb{C} such that if $h\lambda_i$ is inside the region of stability for all i , then $x(\tau_m) \rightarrow 0$ as $m \rightarrow \infty$. The numerical integration method is *A-stable* if the region of stability includes the entire left-half plane of \mathbb{C} . ■

The above definition differs from the original definition given by Dahlquist[42] in that a matrix rather than scalar test problem is used[6]. As will become apparent in following sections, a matrix test problem is more appropriate for analyzing methods designed for large scale systems.

Both the explicit-Euler and implicit-Euler algorithms can be used to produce arbitrarily accurate discrete approximations to the exact solution of Eqn. (2.9), as both are convergent. The implicit-Euler algorithm will allow much larger timesteps to be used with no appreciable loss of accuracy and hence will be more efficient. But improving efficiency is not the only reason one would choose implicit-Euler, or another A-stable numerical integration method. There is also the consideration of numerical robustness. That is, if an A-stable method is used, the timestep can safely be set by *only* considering local truncation error criteria, which can be reasonably estimated. If a method that is not A-stable is used, the timestep must be bounded to insure stability. Such a bound will be a function of the eigenvalues for a linear problem, and it is difficult to get reasonable estimates of eigenvalues.

SECTION 2.2.3 - CHARGE CONSERVATION

Many differential equation systems generated from physical problems can be characterized by the preservation of certain quantities, and frequently it is important that the numerical method also preserve these quantities. For example, when numerically solving the differential equations that describe the motion of a swinging pendulum in a frictionless environment, it is important to insure energy remains constant. If energy increased due to numerical error, the computed solution would indicate that the pendulum would swing higher and higher, and if energy were lost, the computed solution would indicate that the pendulum would eventually come to a halt.

In the case of systems of equations that describe circuits, charge is a physical constant. To show this, consider surrounding arbitrary circuit by a Gaussian surface. Since the surface is unpunctured, the charge contained inside must remain constant[43]. As a consequence, the sum of all the currents must be zero, as the sum of the currents is the derivative with respect to t of the sum of the charge.

This truly trivial observation can not directly apply to the differential equation systems constructed using nodal analysis as above. If the sum of the node charges in Eqn. (2.4) were precisely zero, then $C(x,u)$ in Eqn. (2.2) would be singular and Eqn. (2.2) would not necessarily have a unique solution. In order to produce systems of equations that do have unique solutions, the KCL equations for an arbitrary reference node and for nodes for which the voltages are given *a priori* are not included, and a solution for the reference node of $v_{ref}(t) = 0$ for all t is assumed.

As an example, consider the simple resistor-capacitor circuit of Fig. 2.6a. In terms of charges, the differential equation that describes the behavior of the circuit is

$$\dot{q}(v(t)) = -gv(t) \quad v(0) = 1.0,$$

where the charge $q(v(t)) = cv(t)$. The solution, $v(t) = e^{-\frac{g}{c}t}$, is not a constant, so neither is the charge q . The differential equation does not exhibit charge conservation because not all the charges have been considered, and only the sum remains constant. The charge on the ground or reference node is $-cv(t)$ and obviously the sum of the two is zero for all t .

If KCL is applied to every node in the resistor-capacitor example, including the reference node, an appended system is generated

$$\dot{v}(t) - \dot{v}_{ref}(t) = -\frac{g}{c}(v(t) - v_{ref}(t))$$

$$\dot{v}_{ref}(t) - \dot{v}(t) = -\frac{g}{c}(v_{ref}(t) - v(t))$$

which does not have a unique solution, but an infinite collection, unless it is assumed $v_{ref}(t) = 0$.

However, for any of the solutions the sum of the node charges remains constant.

It is possible to use appended systems generated by applying KCL to every node in a circuit to test how well a numerical integration method conserves charge. If the method is applied to the appended system then charge conservation can be checked by summing all the charges at each timestep to insure the sum remains constant. The algebraic equations generated by the numerical integration method can still be solved in the usual fashion, with the known node voltages and a reference voltage used to eliminate the equations associated with the appended differential equations.

Explicit-Euler applied to an autonomous system (independent of $u(t)$) of the form of Eqn. (2.2) constructed from applying KCL to every node in the circuit yields,

$$\frac{dq}{dv}(\hat{v}(\tau_m))(\hat{v}(\tau_{m+1}) - \hat{v}(\tau_m)) = h_{m+1} f(\hat{v}(\tau_m))$$

where $\frac{dq}{dv}(\hat{v}(\tau_m))$ is the, possibly singular, jacobian of $q(\hat{v}(\tau_m))$, the vector of *all* the node charges. If it is assumed that at τ_m the sum of the node charges $\sum_{i=1}^m q_i(\hat{v}(\tau_m)) = K$, where K is some constant, then charge is conserved only if $\sum_{i=1}^n q_i(\hat{v}(\tau_{m+1}))$ is also equal to K . This is not necessarily the case, as can be seen from the Taylor series expansion of $q(\hat{v}(\tau_{m+1}))$ about $q(\hat{v}(\tau_m))$,

$$\begin{aligned} q(\hat{v}(\tau_{m+1})) &= q(\hat{v}(\tau_m)) + \frac{dq}{dv}(\hat{v}(\tau_m))(\hat{v}(\tau_{m+1}) - \hat{v}(\tau_m)) \\ &\quad + \frac{d^2q}{dv^2}(\hat{v}(\tau))(\hat{v}(\tau_{m+1}) - \hat{v}(\tau_m))(\hat{v}(\tau_{m+1}) - \hat{v}(\tau_m)) \end{aligned} \quad [2.10]$$

where $\hat{v}(\tau) \in [\hat{v}(\tau_m), \hat{v}(\tau_{m+1})]$. Substituting $h_{m+1} f(\hat{v}(\tau_m))$ for $\frac{dq}{dv}(\hat{v}(\tau_m))(\hat{v}(\tau_{m+1}) - \hat{v}(\tau_m))$ leads to

$$q(\hat{v}(\tau_{m+1})) = q(\hat{v}(\tau_m)) + h_{m+1} f(\hat{v}(\tau_m)) +$$

$$\frac{d^2 q}{dv^2}(\hat{v}(\tau))(\hat{v}(\tau_{m+1}) - \hat{v}(\tau_m))(\hat{v}(\tau_{m+1}) - \hat{v}(\tau_m))$$

Summing the node charges,

$$\sum_{i=1}^n q_i(\hat{v}(\tau_{m+1})) = \sum_{i=1}^n q_i(\hat{v}(\tau_m)) + \sum_{i=1}^n h_{m+1} f_i(\hat{v}(\tau_m)) + o(h_{m+1}^2). \quad [2.11]$$

where $o(\cdot)$ is any function such that $\lim_{\alpha \rightarrow 0} \frac{|o(\alpha)|}{\alpha} < \infty$. To simplify Eqn. (2.11), another property of the original network from which the KCL equations were generated can be used. Since $f(\cdot)$ is the vector of sums of the currents incident at each node, and as any current leaving a node must arrive at some other node, $\sum_{i=1}^n f_i(\hat{v}(\tau_m))$ must be identically zero. Using this fact leads to

$$\sum_{i=1}^n q_i(\hat{v}(\tau_{n+1})) = K + O(h_{n+1}^2),$$

which implies that the sum of the node charges will not remain constant unless the second order term in Eqn. (2.10) is zero, which will be true if all the node charges are linear functions of the node voltages, but will not be true in general.

The sum of the charges is constant in the limit as h_{m+1} goes to zero, so the nonconstant charge can be viewed as another measure of the local truncation error. However, if the same integration method is applied slightly differently, using charge as a state variable, then the sum of the node charges will stay constant regardless of the stepsize. To demonstrate this we again apply the explicit-Euler algorithm, but to the system in the form of an autonomous version of Eqn. (2.4). Discretizing the charge function leads to,

$$q(v(\tau_{n+1})) - q(v(\tau_n)) = h_{n+1} f(v(\tau_n)).$$

That the sum of charge is constant, independent of the stepsize, follows from:

$$\sum_{i=1}^n q_i(v(\tau_{m+1})) = \sum_{i=1}^n q_i(v(\tau_m)) + \sum_{i=1}^n h_{m+1} f_i(v(\tau_m))$$

and the fact, mentioned above, that $\sum_{i=1}^n f_i(\hat{v}(\tau_m)) = 0$.

We use these ideas to precisely define the charge conservation property.

Definition 2.5: A system of the form of Eqn. (2.4) is of type S if it has the following two properties: for any exact solution the sum $\sum_{i=1}^n q_i(x(t))$ is a constant independent of t ; and $\sum_{i=1}^n f_i(v) = 0$ for any $v \in \mathbb{R}^n$. A numerical integration method has the *charge conservation* property if when applied to any system of type S , the computed sequence $\{\hat{v}(\tau_i)\}$ is such that $\sum_{i=1}^n q_i(v(\tau_m))$ is a constant independent of m . ■

In section 3.1 we will show that all multistep integration methods applied with charge as the state variable have the charge conservation property.

SECTION 2.2.4 - DOMAIN OF DEPENDENCE

In the area of partial differential equations, the concept of *domain of dependence* is well-known[44]. The idea is that partial differential equations can be characterized by how rapidly the behavior of points in space will propagate with time. As time increases, the space of points that can effect a given point, referred to as the given point's domain of dependence, grows. For a numerical method used to solve the partial differential equation to be convergent, that is to produce arbitrarily accurate solutions as the distance between discretization points becomes small, the numerical method must propagate the behavior of each point in space at a rate that at least approaches the rate of the original partial differential equation. In the language of domain of dependence, a numerical method is convergent only if for each point in space, as the distance between discretization points become small the numerical domain of dependence includes, or comes arbitrarily close to covering, the domain of dependence of the partial differential system.

In this section, an analogous concept will be introduced for large systems of ordinary differential equations. But rather than comparing the domain of dependence of a numerical method to that of the differential equation system to investigate the numerical method's convergence properties, we will show that domain of dependence plays a role in the accuracy of the integration method, and how well the errors due to discretization can be controlled.

Consider the following differential equation system

$$\dot{x}_1(t) = -(x_1(t) - 0.01u(t)) \quad [2.12a]$$

$$\dot{x}_2(t) = -(x_2(t) - 10x_1(t))$$

.

.

$$\dot{x}_n(t) = -(x_n(t) - 10x_{n-1}(t)).$$

$$x_i(0) = 0, \quad i \in \{1, \dots, n\}$$

where the input $u(t) = 1$ for all $t \geq 0$.

The exact solution for this system is:

$$x_i(t) = 10^{i-3} \left[1 - \left(\sum_{j=0}^{i-1} \frac{t^j}{j!} \right) \right] e^{-t}. \quad [2.12b]$$

As can be seen by examining Eqn. (2.12b), the solution to the system of Eqn. (2.12a) is a propagating step that is being smoothed and is growing rapidly in amplitude through n stages. Systems with this type of behavior are extremely common among circuit examples (a chain of inverters, for example).

If the explicit-Euler algorithm is applied to Eqn. (2.12a), the computed value for $\hat{x}_1(\tau_1) = 0.01h_1$ and $\hat{x}_i(\tau_1) = 0$ for all $1 < i \leq n$. In fact, \hat{x}_i will remain zero until the i^{th} timestep regardless of the size of the timestep. This slow propagation of the solution introduces an error that is in the form of a delay, that is $\hat{x}_i(\tau_j)$ does not change until $j \geq i$. Explicit-Euler is convergent, so this delay error *in time* must be driven to zero as the timestep decreases, and it does, because τ_j approaches zero.

If implicit-Euler is applied to Eqn (2.12a), then $x_i(\tau_1) = \frac{10^{-3}h_1}{(1+h_1)^i}$. Therefore, when the implicit-Euler algorithm is used, the behavior of the input is propagated throughout the entire system in one timestep and there is no error due to delayed propagation of information. This does *not* necessarily imply that implicit-Euler is more accurate than the explicit-Euler algorithm. For example, applied to Eqn. (2.12a) with a timestep $h_1 = 1$, explicit-Euler produces the solution $\hat{x}_5(\tau_1) = 0.0$, while implicit-Euler produces the solution $\hat{x}_5(\tau_1) = 3.125$. The exact solution is $x_5(1) = 0.359$, so in this case, the explicit-Euler computed solution is closer to the exact solution than the implicit-Euler computed solution, though neither method produces very accurate results.

For this example, accuracy clearly isn't the reason for preferring the implicit-Euler's rapid propagation of information to explicit-Euler. Implicit-Euler is a more reliable integration method for this example because the error due to discretization in the computed solution is *more visible* than the discretization error in the computed solution produced by the explicit-Euler algorithm. To see why this is the case, consider the local truncation error estimate presented in Section 2.2.1,

$$LTE \approx h_{m+1}^2 \frac{\frac{\hat{x}(\tau_{m+1}) - \hat{x}(\tau_m)}{h_{m+1}} - \frac{\hat{x}(\tau_m) - \hat{x}(\tau_{m-1})}{h_m}}{(h_{m+1} + h_m)}. \quad [2.13]$$

Since in this case, $m = 0$, $\hat{x}(\tau_m) = \hat{x}(\tau_{m-1}) = x(0)$ and $h_m = 0$, Eqn. (2.13) can be simplified to

$$LTE \approx h_{m+1}^2 (\hat{x}(\tau_1) - x(0))$$

For explicit-Euler this estimate indicates that the LTE for $x_5(\tau_1)$ is zero, which is a severe underestimate. A timestep control scheme based on local truncation error would not shrink the timestep in this case, and a very inaccurate solution would be computed. For the implicit-Euler algorithm, the error estimate is 3.125 which is larger than the actual LTE, but this is safe, because an LTE-based timestep control scheme will detect the error and reduce the timestep.

This example indicates that when applying the explicit-Euler algorithm to a large system, a timestep dependent limit is introduced on how fast the behavior of an individual state variables propagate through the system. The delay error due to this limited rate of propagation is different from a local truncation error. An arbitrarily high order explicit multistep integration method could have been used at each step, and still $x_i(\tau_m)$ would have been zero until the i^{th} timestep. The implicit-Euler algorithm does not introduce such an *a priori* limitation on how fast the behavior of an individual state variables propagate through the system. Because of this, when the system behavior is faster than can be propagated by the explicit-Euler algorithm, the implicit-Euler algorithm can produce more accurate results, but more importantly, when it produces results that are in error, those errors are more observable.

We end this section, and this chapter, by connecting the concept of the delay introduced by an integration method, the *numerical delay* to that of *Domain of Dependence*, the concept borrowed from the study of partial differential equations. This connection will provide a simple tool for testing integration methods to determine for what type of systems they will introduce numerical delay.

For this purpose, we can define the numerical delay as follows:

Definition 2.6: Given a numerical integration method applied to a system of the form $\dot{x}(t) = Ax(t)$ with some initial condition $x(0) = x_0$, if $x_i(\tau) - x_i(0) \neq 0$ for all $\tau \in (0, \tilde{\tau}]$ for some $\tilde{\tau} > 0$, then the *numerical delay to the i^{th} variable* is defined as the smallest integer M_i such that $\hat{x}(\tau_{M_i+1}) - x_i(0) \neq 0$. If no such $\tilde{\tau}$ exists, the numerical delay to the i^{th} variable, M_i , is zero. The *numerical delay* for the integration method applied to the given system with the given initial condition is the maximum over all i of the M_i ■

In the example above, the numerical delay for implicit-Euler algorithm applied to Eqn. (2.12a) is zero, and the numerical delay for explicit-Euler is $n - 1$.

The description of the role of domain of dependence will be based on the following general definition:

Definition 2.7: Given an equation of the form $y = f(x)$, where $x, y \in \mathbb{R}^n$, and $f: \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$, the domain of dependence of the j^{th} variable of the vector y , y_j , is the set of all x_i , $i \in \{1, \dots, n\}$ such that for some x , $\frac{\partial f_j}{\partial x_i} \neq 0$ ■.

Given the matrix test problem

$$\dot{x}(t) = Ax(t) \quad x(0) = x_0 \quad [2.14]$$

where $x(t) \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$, the exact solution at $t = h$ is, in series form,

$$x(h) = [I + hA + \frac{h^2}{2}A^2 + \frac{h^3}{6}A^3 + \dots]x(0). \quad [2.15]$$

The domain of dependence of $x_i(h)$ can be deduced directly from Eqn. (2.15). The variable $x_j(0)$ is in the domain of dependence of $x_i(h)$ if the i, j^{th} element of A^n is nonzero for some n .

The equation for one step of explicit-Euler applied to Eqn. (2.15) is

$$\hat{x}(\tau_1) = [I + h_1A]x(0). \quad [2.16]$$

As can be seen from the equation, the domain of dependence for the x_i^{th} variable in Eqn. (2.16) will be a proper subset of the domain of dependence for the x_i^{th} variable in Eqn. (2.15) unless the powers of the matrix A do not add additional nonzero terms. This would occur, for example, in the case where A is diagonal. If instead, one step of implicit-Euler were applied to Eqn. (2.14), the following series expansion results:

$$\hat{x}(\tau_1) = [I + hA + h^2A^2 + h^3A^3 + \dots]x(0), \quad [2.17]$$

where the series expansion is valid for h such that $h\rho < 1$ where ρ is the spectral radius of A . Comparing Eqn. (2.17) to Eqn. (2.15), it can be seen that for a small enough h the domains of dependence of the exact solution and the implicit-Euler algorithm are *identical* for each variable, $x_j(h)$. We define this property below as *exhaustive domain of dependence*.

Definition 2.8: If the domain of dependence of each element of the vector produced by one step of an integration method applied to Eqn. (2.14) matches the domain of dependence of the corresponding element in the left hand side vector of Eqn. (2.15) for a small enough timestep h and for any A and any initial condition x_0 , then the numerical method is said to have an *exhaustive domain of dependence*. ■

The following theorem relating domain of dependence to numerical delay follows directly from the definitions:

Theorem 2.2: If a numerical integration method has an exhaustive domain of dependence then the numerical delay of the integration method is zero for any A and any x_0 . ■

If one step of a numerical method has a smaller domain of dependence than the original differential equation, then a numerical delay will be introduced and the timesteps used for the calculation will have to be bounded to insure rapid enough propagation of variable behavior. Like bounds on the timestep to insure stability for non-A-stable methods, this additional constraint is difficult to estimate, and must be done very conservatively. The explicit-Euler example above demonstrates how difficult the error is to even observe, because the effected variables, for which the error occurs, are left unperturbed. For this reason, a robust numerical integration algorithm for large systems must either use a method like implicit-Euler, which has an exhaustive domain of dependence, or have some technique for checking that system variables have propagated far enough.

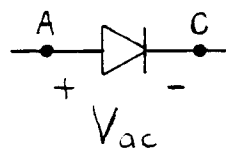


Figure 2.1 - A Diode in Free Space

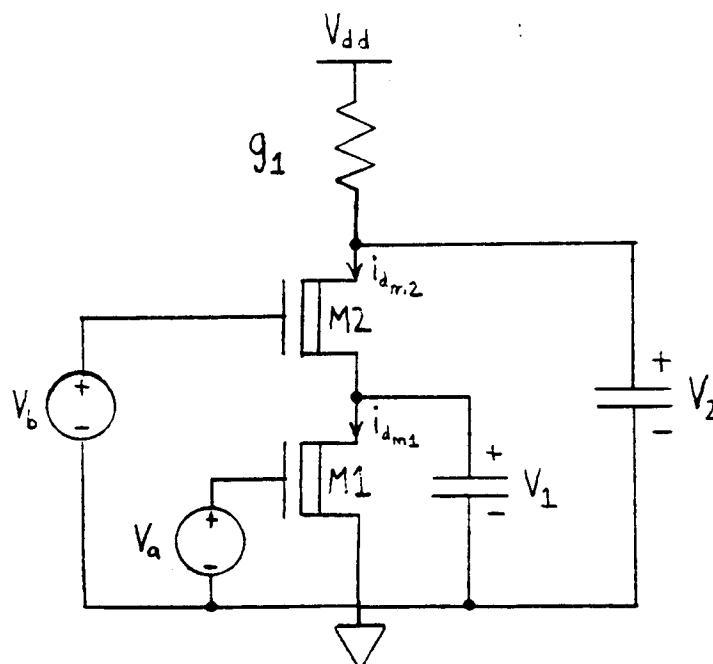


Figure 2.2 - An MOS Nand Gate

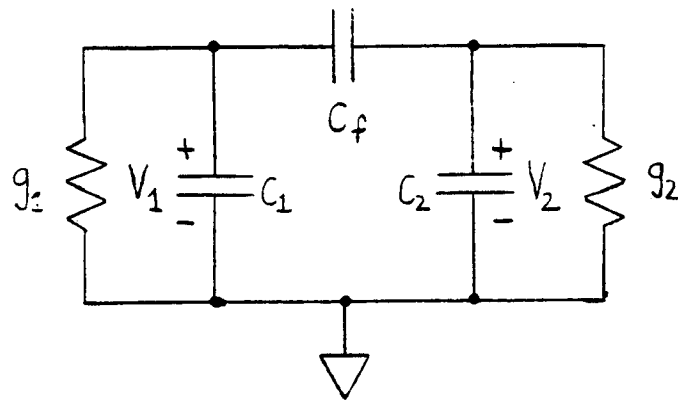


Figure 2.3 - Floating Capacitor Example

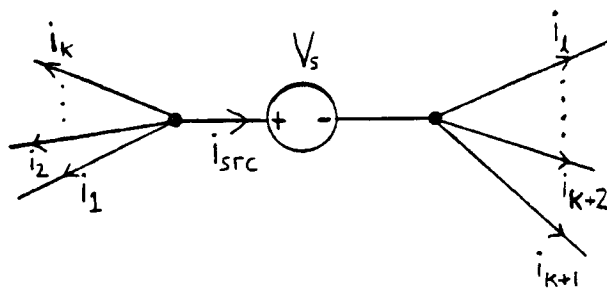


Figure 2.4 - Floating Voltage Source

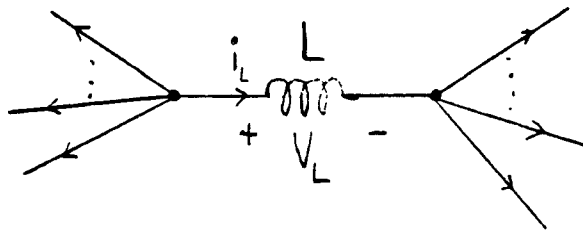


Figure 2.5a - Floating Inductor Example

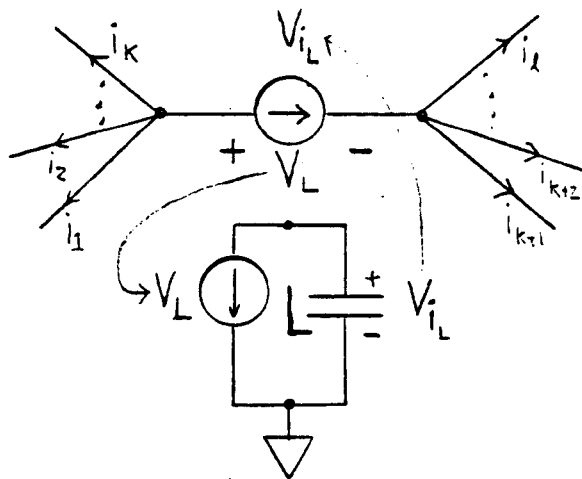


Figure 2.5b - Floating Inductor Equivalent Circuit

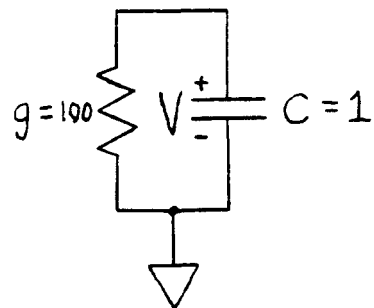


Figure 2.6 - Stiff Resistor-Capacitor Circuit

CHAPTER 3 - NUMERICAL TECHNIQUES

The implicit multistep integration algorithms used in general purpose circuit simulation programs like SPICE2 [2] and ASTAP[3] have proved to be extremely reliable, but are computationally expensive when applied to large systems. This is because each step of the numerical integration requires the solution of a large implicit nonlinear algebraic system. Two approaches have been used to reduce the computation time required by these methods. Decomposition techniques have been applied to improve the efficiency of the solution of the large algebraic systems generated by implicit integration methods, and less computationally demanding semi-implicit numerical integration algorithms have been developed. In this chapter we will start by demonstrating that the implicit multistep integration algorithms used in general purpose circuit simulation programs have the three key properties described in Chapter 2, *charge conservation*, *exhaustive domain of dependence* and *stiff stability*. Following, the relaxation algorithms that have been used in circuit simulators for solving the large nonlinear algebraic systems generated by implicit integration methods will be described. Then the semi-implicit integration methods used in special purpose programs like MOTIS[7], MOTIS2[8], and SPLICE[45] will then be analyzed with respect to their domain of dependence and stability properties. Finally, we will end this chapter by comparing some of the special purpose integration algorithms with algebraic relaxation methods.

SECTION 3.1 - NUMERICAL INTEGRATION IN GENERAL-PURPOSE SIMULATORS

Most of the general-purpose circuit simulation programs use implicit multistep integration algorithms applied to the state variable charge (and if inductances are included, also fluxes). That is, given a system of the form

$$\dot{q}(x(t), u(t)) = f(x(t), u(t)), \quad [3.1]$$

where $x(t) \in \mathbb{R}^n$ is the system state, usually the vector of node voltages appended by inductor currents, $u(t) \in \mathbb{R}^l$, is the vector of inputs, and is continuously differentiable with respect to t , $f: \mathbb{R}^n \times \mathbb{R}^l \rightarrow \mathbb{R}^n$, continuously differentiable, is usually the vector of sums of currents entering a node, and $q: \mathbb{R}^n \times \mathbb{R}^l \rightarrow \mathbb{R}^n$, continuously differentiable, is usually the vector of node charges or fluxes. A function, \hat{f} is defined such that $\hat{f}(q(x(t)), u(t)) = f(x(t), u(t))$. Using such an \hat{f} , Eqn. (3.1) is converted to a system in normal form,

$$\dot{q}(x(t), u(t)) = \hat{f}(q(x(t)), u(t)). \quad [3.2]$$

One of the collection of multistep integration methods is then used to solve Eqn. (3.2). The general form for a multistep integration method applied to Eqn. (3.2) is

$$\sum_{i=0}^k \alpha_i q(\hat{x}(\tau_{m-i}), u(\tau_{m-i})) = h_m \sum_{i=0}^l \beta_i \hat{f}(q(\hat{x}(\tau_{m-i}), u(\tau_{m-i}))) \quad [3.3]$$

which is identical to

$$\sum_{i=0}^k \alpha_i q(\hat{x}(\tau_{m-i}), u(\tau_{m-i})) = h_m \sum_{i=0}^l \beta_i \hat{f}(\hat{x}(\tau_{m-i}), u(\tau_{m-i})) \quad [3.4]$$

where k, l are positive integers, $\alpha_0 = 1$, and $\alpha_i, \beta_j \in \mathbb{R}$ for $0 < i \leq k, 0 \leq j \leq l$ depend on the integration method and the ratio of the timesteps h_i , $m - \max(k, l) \leq i \leq m$. For example, the fixed-timestep explicit-Euler algorithm used for examples in Chapter 2 can be derived from Eqn. (3.4) by setting $k = 1, l = 1, \alpha_0 = 1, \alpha_1 = -1, \beta_0 = 0$, and $\beta_1 = 1$. To derive implicit-Euler the coefficients remain the same except $\beta_0 = 1$, and $\beta_1 = 0$.

Not all collections of α 's and β 's produce useful numerical integration methods. Consistency is one limitation on the choice of coefficients. It is well known that for a multistep method to be consistent,

$$\sum_{i=1}^k \alpha_i = -1$$

and

$$\sum_{i=1}^k i \alpha_i + \sum_{i=0}^l \beta_i = 0$$

where it is assumed that $\alpha_0 = 1[1]$. In addition, if $\beta_0 = 0$ the integration method is said to be *explicit*, otherwise, the method is *implicit*.

When a multistep method is applied to a system of the form of Eqn. (3.2), the state at the m^{th} step, $\hat{x}(\tau_m)$, is computed by solving

$$q(\hat{x}(\tau_m), u(\tau_m)) + h_m \beta_0 f(\hat{x}(\tau_m), u(\tau_m)) + \quad [3.5]$$

$$\sum_{i=1}^k \alpha_i q(\hat{x}(\tau_{m-i}), u(\tau_{m-i})) - h_m \sum_{i=1}^l \beta_i f(\hat{x}(\tau_{m-i}), u(\tau_{m-i})) = 0.$$

for $\hat{x}(\tau_m)$ given $\hat{x}(\tau_j)$, $q(\hat{x}(\tau_j), u(\tau_j))$, and $f(\hat{x}(\tau_j), u(\tau_j))$ for all $j < m$.

Implicit nonlinear algebraic systems generated by integration methods are usually solved using the iterative Newton-Raphson(NR) method. The NR algorithm is used because it is guaranteed to converge if the initial guess is close enough to the exact solution. From this observation it follows that as the exact solution to the differential equation is a continuous function, it is possible to pick a timestep small enough to insure the NR algorithm will converge. Also, the NR algorithm will converge independent of the stiffness of the system, which follows from the observation that the NR algorithm will solve a linear problem exactly in one step.

The general Newton-Raphson iteration equation to solve $F(x) = 0$ where $x \in \mathbb{R}^n$ and $F: \mathbb{R}^n \rightarrow \mathbb{R}^n$ is

$$J_F(x^k)(x^k - x^{k-1}) = -F(x^{k-1}) \quad [3.6]$$

where J_F is the jacobian of F with respect to x . The iteration is continued until $\|x^k - x^{k-1}\| < \epsilon$ and $F(x^k)$ is close enough to 0. If the Newton algorithm is used to solve Eqn. (3.5) for $x(\tau_m)$, the residue at the k^{th} step, $F(x^k(\tau_m))$, is

$$F(x^k(\tau_m)) = q(\hat{x}^k(\tau_m), u(\tau_m)) + \beta_0 f(\hat{x}^k(\tau_m), u(\tau_m)) + \quad [3.7]$$

$$\sum_{i=1}^k \alpha_i q(\hat{x}(\tau_{m-i}), u(\tau_{m-i})) - h_m \sum_{i=1}^l \beta_i f(\hat{x}(\tau_{m-i}), u(\tau_{m-i})).$$

and the Jacobian $J_F(x^k(\tau_m))$ is

$$J_F(x^k(\tau_m)) = \frac{\partial q}{\partial x}(\hat{x}^k(\tau_m), u(\tau_m)) - \beta_0 \frac{\partial f}{\partial x}(\hat{x}^k(\tau_m), u(\tau_m)) \quad [3.8]$$

Then $\hat{x}^{k+1}(\tau_m)$ is derived from $\hat{x}^k(\tau_m)$ by solving the linear system of equations

$$J_F(\hat{x}^k(\tau_m)) [\hat{x}^{k+1}(\tau_m) - \hat{x}^k(\tau_m)] = -F(\hat{x}^k(\tau_m)) \quad [3.9]$$

The Newton iteration is continued until sufficient convergence is achieved, that is $\|\hat{x}^{k+1}(\tau_m) - \hat{x}^k(\tau_m)\| < \epsilon$ and $F(\hat{x}^k(\tau_m))$ is close enough to zero.

Note that here, even if the integration algorithm is explicit ($\beta_0 = 0$), Eqn. (3.5) will still be an implicit algebraic problem with respect to $\hat{x}(\tau_m)$. This occurs because the multistep algorithm was applied using charge as a state variable, and charge is a nonlinear function of x .

One of the important reasons for applying the integration method to the system in the form of Eqn. (3.2) is that the charge conservation property of Definition 2.5 holds for any consistent multi-step method.

Theorem 3.1: Any consistent multistep method of the form of Eqn. (3.3) has the charge conservation property. ■

Proof of Theorem 3.1

Let the system of Eqn. (3.1) be of type S , as given in Definition 2.5. To show charge conservation, the vector elements in Eqn. (3.4) are summed to form

$$\sum_{i=1}^n \sum_{j=0}^k \alpha_j q_i(\hat{x}(\tau_{m-j}), u(\tau_{m-j})) = \sum_{i=1}^n h_m \sum_{j=0}^l \beta_j f_i(\hat{x}(\tau_{m-j}), u(\tau_{m-j})). \quad [3.8]$$

Interchanging summations yields

$$\sum_{j=0}^k \alpha_j \sum_{i=1}^n q_i(\hat{x}(\tau_{m-j}), u(\tau_{m-j})) = h_m \sum_{j=0}^l \beta_j \sum_{i=1}^n f_i(\hat{x}(\tau_{m-j}), u(\tau_{m-j})). \quad [3.9]$$

Since the original system is of type S , $\sum_{i=1}^n f_i(\hat{x}(\tau_{m-j}), u(\tau_{m-j})) = 0$. Substituting into Eqn. (3.9) and using that $\alpha_0 = 1$,

$$\sum_{i=1}^n q_i(\hat{x}(\tau_m), u(\tau_m)) = - \left[\sum_{j=1}^k \alpha_j \sum_{i=1}^n q_i(\hat{x}(\tau_{m-j}), u(\tau_{m-j})) \right]. \quad [3.10]$$

Assuming that charge has been conserved up to the m^{th} step $\sum_{i=1}^n q_i(\hat{x}(\tau_j)) = K$ for $j < m$. Then as

$\sum_{j=1}^k \alpha_j = -1$ because the method is assumed consistent,

$$\sum_{i=1}^n q_i(\hat{x}(\tau_m), u(\tau_m)) = K, \quad [3.11]$$

which proves the theorem ■.

Exhaustive domain of dependence is also easy to show for most *implicit* multistep methods.

Theorem 3.2: Any implicit multistep method with $\alpha_1 \neq 0$ has an exhaustive domain of dependence when applied to a system of the form $\dot{x}(t) = Ax(t)$, where $x(t) \in \mathbb{R}^n$, and $A \in \mathbb{R}^{n \times n}$. ■

Proof of Theorem 3.2

The general form for a multistep method applied to $\dot{x}(t) = Ax(t)$ is

$$\sum_{i=0}^k \alpha_i \hat{x}(\tau_{m-i}) = h_m \sum_{i=0}^l \beta_i A \hat{x}(\tau_{m-i}). \quad [3.12]$$

Reorganizing and using the fact that $\alpha_0 = 1$,

$$\hat{x}(\tau_m) = [I - h_m \beta_0 A]^{-1} \left[\sum_{i=1}^k \alpha_i \hat{x}(\tau_{m-i}) + h_m \sum_{i=1}^l \beta_i A \hat{x}(\tau_{m-i}) \right].$$

Since the method is implicit, $\beta_0 \neq 0$, and for small h_m $[I - h_m \beta_0 A]^{-1}$ can be expanded to yield:

$$\hat{x}(\tau_m) = [I + h_m \beta_0 A + \frac{1}{2}(h_m \beta_0 A)^2 + \frac{1}{6}(h_m \beta_0 A)^3 + \dots] [\alpha_1 - \beta_1 A] \hat{x}(\tau_{m-1}) + \quad [3.13]$$

$$[I - h_m \beta_0 A]^{-1} \left[\sum_{i=2}^k \alpha_i \hat{x}(\tau_{m-i}) + h_m \sum_{i=2}^l \beta_i A \hat{x}(\tau_{m-i}) \right].$$

Following the same argument as presented in Section 2.2.4, the variable $\hat{x}_i(\tau_{m-1})$ is in the domain of dependence of $\hat{x}_j(\tau_m)$ if the i,j^{th} element of A^n is nonzero for some n , which matches the differential equation and therefore proves the theorem. ■

The general question of the region of stability for multistep integration methods has received considerable attention[1,42,46] and the wealth of material on this question will not be reproduced here. Instead, we will mention the results that are most critical for circuit simulation applications. Perhaps the most important result is that there are no A-stable multistep integration methods whose local truncation error is of order higher than h^3 . This is known as the Dahlquist barrier[42]. For this reason, the program SPICE[2] uses a combination of the implicit-Euler mentioned in Chapter 2 and the trapezoidal rule (corresponding to $\alpha_0 = 1, \alpha_1 = -1, \beta_0 = 0.5, \beta_1 = 0.5$) and as a user option, can also use the variable-order (up to six) backward-difference methods[1]. The program ASTAP[3] uses the variable-order backward-difference methods. The first and second order backward-difference methods are A-stable, but the higher order backward-difference integration methods are only *stiffly stable*. By this, it is meant that the region of stability for these methods include the real line in the open left-half plane of z and some sections in the open left-half plane about the real line[1].

SECTION 3.2 - RELAXATION DECOMPOSITION

As mentioned above, the implicit multistep integration methods used in all the general-purpose circuit simulation programs require solving an implicit system of nonlinear algebraic equations at each timestep. The algebraic system is usually cast into the form $F(x) = 0$ where $F: \mathbb{R}^n \rightarrow \mathbb{R}^n$, and $x \in \mathbb{R}^n$, which is then solved using the iterative Newton-Raphson(NR) algorithm as in Eqn. (3.6).

The computation of the Newton iterates can be viewed as two pieces, evaluating the function F , and its Jacobian J_F , and performing a matrix solution. The computational cost of performing the matrix solution grows superlinearly with the size of the problem, as n^α , where n the number of equations in the system and $\alpha > 1$. Circuit simulation programs are intended to handle large circuits,

and as the Jacobian matrices are sparse, sparse matrix techniques[40] are used to keep α as low as possible. It has been empirically observed that the time to perform a sparse matrix solution grows as $1.2 < \alpha < 1.4$ for the matrices associated with circuit simulation problems. The computational cost of a function evaluation grows linearly with the size of problem, but for circuit simulation problems, the evaluation of F and J_F is a complicated task. For each element (transistor, capacitor, resistor, etc) in the circuit, the currents, the charges and their derivatives must be evaluated. For example, the evaluation of the currents and charges associated with one MOS transistor requires more than a hundred floating point operations.

Because the computation involved in calculating each transistor's charge and current characteristic is much more complicated than the simpler operations involved in the matrix solution, for small to medium sized problems the function evaluation time dominates the sparse matrix solution time. It is only when the problem involves more than several thousand equations that the matrix solution time dominates. For this reason, the most useful decomposition techniques applied to circuit simulation problems reduce both the matrix solution time and the function evaluation time.

Two approaches to decomposition have been used in circuit simulation programs. The first, which we will not describe in detail here, is referred to as tearing decomposition. For linear equations, tearing is a form of Block LU Factorization[4, 5, 47, 48, 49, 50]. Its application to nonlinear systems has led to Multi-level Newton algorithms[52]. The second approach, closer to the focus of this thesis, has been to apply the various forms of the iterative relaxation-Newton or SOR-Newton algorithms[21, 53].

As background for the relaxation-Newton algorithm, we will present an extremely brief description of the Gauss-Jacobi and Gauss-Seidel relaxation methods starting with the algorithms for linear systems. A complete discussion can be found in [28].

The linear problem $Ax - b = 0$ where $x = (x^1, \dots, x_n)^T$, $b = (b^1, \dots, b_n)^T$, $x_i, b_i \in \mathbb{R}$, and $A = (a_{ij})$, $A \in \mathbb{R}^{n \times n}$ can be solved exactly using gaussian elimination (with pivoting) given A is nonsingular. For matrices with certain properties, it is also possible to solve for x in an iterative

fashion, where each step of the iteration involves inverting a sequence of one-dimensional problems.

For example, there is the Gauss-Jacobi relaxation algorithm

Algorithm 3.1 (Gauss-Jacobi Algorithm for solving $Ax = b$)

The superscript k is the iteration count and ϵ is a small positive number.

$k \leftarrow 0$;

Guess some x^0 .

repeat {

$k \leftarrow k + 1$

foreach ($i \in \{1, \dots, n\}$) $x_i^k = \frac{1}{a_{ii}} [b_i - (\sum_{j=1}^{i-1} a_{ij}x_j^{k-1} + \sum_{j=i+1}^n a_{ij}x_j^{k-1})]$

} until ($\|x^k - x^{k-1}\| \leq \epsilon$)

■

The Gauss-Seidel relaxation algorithm is very similar, and can be generated from Algorithm 3.1 by

altering the update equation for x_i^k to

$$x_i^k = \frac{1}{a_{ii}} [b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k + \sum_{j=i+1}^n a_{ij}x_j^{k-1}].$$

The Gauss-Jacobi algorithm can be written in matrix form as

$$Dx^k + (L + U)x^{k-1} = b$$

and the Gauss-Seidel algorithm can be written in matrix form as

$$(L + D)x^k + Ux^{k-1} = b$$

where $L, D, U \in \mathbb{R}^{n \times n}$ are strictly lower triangular, diagonal, and strictly upper triangular respectively,

and are such that $A = L + D + U$. Taking the difference between k and $k - 1$ iteration we get

$$x^k - x^{k-1} = D^{-1}(L + U)(x^{k-1} - x^k),$$

for Gauss-Jacobi, and

$$x^k - x^{k-1} = (L + D)^{-1}U(x^{k-1} - x^k),$$

for Gauss-Seidel. It follows that the Gauss-Jacobi relaxation algorithm will converge if the spectral radius of $D^{-1}(L + U)$ is inside the unit circle and Gauss-Seidel relaxation algorithm will converge if the spectral radius of $(L + D)^{-1}U$ is inside the unit circle. This will be true, for example, if A is strictly diagonally dominant [28].

Now consider using the Gauss-Seidel and Gauss-Jacobi relaxation algorithms to solve the nonlinear system $F(x) = 0$ where $F(x) = (f_1(x), \dots, f_n(x))^T$, and $f_i: \mathbb{R}^n \rightarrow \mathbb{R}$. At each step of the relaxation, the x_i element is updated by solving the implicit algebraic equation,

$$f_i(x_1^k, \dots, x_{i-1}^k, x_i^{k+1}, x_{i+1}^k, \dots, x_n^k) = 0. \quad [3.15a]$$

for the Gauss-Jacobi, and

$$f_i(x_1^{k+1}, \dots, x_i^{k+1}, x_{i+1}^k, \dots, x_n^k) = 0. \quad [3.15b]$$

for Gauss-Seidel.

It is possible to use the Newton-Raphson algorithm to accurately solve the implicit algebraic systems of Eqn. (3.15a) and Eqn. (3.15b) at each step, but this is not essential. That is, it has been shown that the asymptotic rate of convergence of the nonlinear relaxation is not reduced if rather than solving the implicit algebraic systems at each step, only one iteration of the Newton method is used[21]. The algorithms so generated are referred to as the relaxation-Newton methods. The Gauss-Jacobi-Newton algorithm for solving systems of the form of Eqn. (3.14) is

$$x_i^{k+1} = x_i^k - \frac{\partial f_i(x^k)}{\partial x_i}^{-1} f_i(x^k) \quad [3.16a]$$

and the Gauss-Seidel-Newton algorithm is

$$x_i^{k+1} = x_i^k - \frac{\partial f_i(x^{k,j})}{\partial x_i} - f_i(x^{k+1,j}) \quad [3.16b]$$

where $x^{k+1,j} = (x_1^{k+1}, \dots, x_{i-1}^{k+1}, x_i^k, \dots, x_n^k)^T$.

There is the following general theorem about the local convergence of relaxation-Newton methods.

Theorem 3.3: If a given $F: \mathbb{R}^n \rightarrow \mathbb{R}^n$ is continuously differentiable, and if there exists an $x \in \mathbb{R}^n$ such that $F(x) = 0$, then if the Jacobian of F at x , $J_F(x)$, is strictly diagonally dominant there exist some $\delta > 0$ such that both the Gauss-Jacobi-Newton or the Gauss-Seidel-Newton iterations applied to F will converge for any x^0 for which $\|x_0 - x\| \leq \delta$. ■

The proof of the above well-known theorem can be found in the references[21]. As a direct consequence, we have the following theorem for the nonlinear algebraic systems generated by consistent multistep integration methods.

Theorem 3.4: Let the Gauss-Seidel-Newton or Gauss-Jacobi-Newton relaxation algorithm be used to solve for $\hat{x}(\tau_m)$ in Eqn. (3.5). If $f(x, u)$ is continuously differentiable, $\frac{\partial q}{\partial x}(x, u)$ is strictly diagonally dominant uniformly over all x , and $\hat{x}(\tau_{m-1})$ is used as the starting point for the relaxation, then there exists an \tilde{h} such that for all $h_m \leq \tilde{h}$ the relaxation will converge to the solution of Eqn. (3.5). ■

As an intuitive explanation for why Theorem 3.4 should be true, and why nonconvergence should ever occur, consider implicit-Euler applied to Eqn. (2.2) with $C(x, u) = C$, where C is strictly diagonally dominant.

$$Cx(\tau_m) = Cx(\tau_{m-1}) + h_m f(x(\tau_m), u(\tau_m)). \quad [3.17]$$

In the limit as $h_m \rightarrow \infty$, Eqn. (3.17) becomes equivalent to solving $f(x(\tau_m), u(\tau_m)) = 0$ for $x(\tau_m)$ by relaxation. Since little is assumed about f other than Lipschitz continuity, it is unlikely that this problem can be solved, in general, with a relaxation method. However, in the limit as the timestep becomes small, Eqn. (3.17) becomes

$$Cx(\tau_m) = b$$

where $b = Cx(\tau_{m-1})$. This problem can be solved by relaxation because C is strictly diagonally dominant. We formalize this observation in the proof of Theorem 3.4.

Proof of Theorem 3.4:

It is sufficient to show that the system of Eqn. (3.5) will satisfy the conditions of Theorem 3.3 for small enough h_m . The Jacobian for the function defined by Eqn. (3.5), J_F , is given in Eqn. (3.8). That J_F is strictly diagonally dominant for h_m small enough follows directly from the observation that in the limit as $h_m \rightarrow 0$, J_F approaches $\frac{\partial q}{\partial x}$ which is a strictly diagonally dominant matrix by assumption. That $\hat{x}(\tau_{m-1})$ is close enough to the solution of Eqn. (3.5) for a small enough h_m follows from the assumption that the multistep method is consistent. Consistency implies $\hat{x}(\tau_m) - \hat{x}(\tau_{m-1})$ is a solution to Eqn. (3.5) for $h_m = 0$ and from the Lipschitz continuity of q and f which imply that $\hat{x}(\tau_m)$ is a continuous function of h_m .

The relaxation-Newton methods have become popular for solving circuit simulation problems for two reasons. The first is that, as mentioned in Chapter 2, for a broad class of circuits the capacitance matrix is diagonally dominant and therefore the relaxation-Newton algorithms are guaranteed to converge if the timestep is made small enough. They are unlike the standard NR methods in that the timestep required is not truly independent of the problem stiffness, an issue which will be presented more thoroughly at the end of this chapter. The second reason for the popularity of the relaxation-Newton methods is that with proper application, it is possible to both avoid matrix solutions and reduce the computation involved in function evaluation. As the system Jacobian is sparse,

the i^{th} component of the function F defined in equation 3.5, F_i , will be a function of only a few components of the vector x . During the relaxation-Newton process this sparsity can be exploited by noting whether or not the components of x on which F_i depends have changed significantly, and if none of them have, not reevaluating F_i . In addition, if F_i is close enough to 0, x_i^{k+1} will be equal to x_i^k and need not be recomputed.

If implemented as described above, such a partial evaluation scheme involves substantial checking, to see if F_i should be reevaluated. This checking can overwhelm the savings due to partial function evaluation. To avoid this, practical relaxation-Newton algorithms are implemented using a *selective trace* technique[33] that simultaneously determines the order in which the relaxation equations are solved and the portion of the function that must be recomputed.

SECTION 3.3 - SEMI-IMPLICIT NUMERICAL INTEGRATION METHODS

Although certain implicit multistep integration methods have all the desirable properties described in Chapter 2, they are computationally expensive when applied to very large systems partly because each timepoint requires a large matrix solution. Semi-implicit integration methods, as the name implies, are constructed to be as implicit as possible without making it necessary to perform standard matrix solutions to compute the timepoints. In this section we will discuss three semi-implicit methods, all of which have been used in circuit simulation applications. In order to simplify the presentation of these algorithms, they will be considered as applied to the following test problem,

$$\dot{x}(t) = Ax(t) \quad x(0) = x_0 \quad [3.18]$$

where $x(t) \in \mathbb{R}^n$, and $A \in \mathbb{R}^{n \times n}$. The properties of these algorithms with respect to domain of dependence and stiff-stability will be considered. This test problem is too simple to indicate the integration methods' charge conservation properties, and that issue will not be considered.

The simplest of the semi-implicit methods is the following mixture of explicit and implicit-Euler[5,7].

$$x(\tau_m) = x(\tau_{m-1}) + h_m [Dx(\tau_m) + (L + U)x(\tau_{m-1})] \quad [3.19]$$

where $L, D, U \in \mathbb{R}^{n \times n}$ are strictly lower triangular, diagonal, and strictly upper triangular respectively, and are such that $A = L + D + U$. Note that this algorithm is identical to solving the algebraic equations generated by implicit-Euler applied to Eqn. (3.18) with with one iteration of a Gauss-Jacobi relaxation scheme, and therefore the algorithm is referred to as the Jacobi-semi-implicit method. Solving for $x(\tau_m)$ leads to

$$x(\tau_m) = (I - h_m D)^{-1} [I + h_m (L + U)] x(\tau_{m-1}). \quad [3.20]$$

Since $(I - h_m D)$ is diagonal, its inverse, if it exists, can be computed trivially. In addition, we have the following stability result (See [6] for similar results).

Theorem 3.5: If the matrix A in Eqn. (3.18) is diagonally dominant with negative diagonal entries, or A is lower or upper triangular, then the region of stability for the Jacobi-semi-implicit method is the open left-half plane of \mathbb{C} . ■

This theorem is of practical value because the systems of differential equations that describe circuits with resistors and grounded capacitors will be of the form of Eqn. (3.18) and will have the diagonal dominance property.

Proof of Theorem 3.5:

To prove the first part of the theorem it is sufficient to show that the matrix M defined by

$$M = (I - h_m D)^{-1} [I + h_m (L + U)] \quad [3.22]$$

has a spectral radius $\rho(M) < 1$ if A is diagonally dominant and has negative diagonal entries, or if A is upper or lower triangular and has its eigenvalues in the open left-half plane of \mathbb{C} . If A is upper or lower triangular, the eigenvalues of A are the diagonal entries, which must be negative by assumption. If A is triangular, M will be triangular, and the eigenvalues of M will be its diagonal entries. The i^{th} diagonal entry of M can be calculated explicitly, and is $\frac{1}{1 + h_m |a_{ii}|}$ which is less than 1. To prove the theorem for the case where A is diagonally dominant and has negative diagonal entries, we use the fact that the spectral radius is bounded by any induced norm. In particular, $\rho(M) \leq \|M\|_\infty = \max_i \sum_{j=1}^n |m_{ij}|$, which can be calculated from

$$\sum_{j=1}^n |m_{ij}| = \frac{\sum_{j=1, j \neq i}^n |a_{ij}|}{\frac{1}{h_m} + |a_{ii}|}$$

and is less than 1 by the diagonal dominance property of A . Therefore, the eigenvalues of M are less than one. ■

Although the stability of the Jacobi-semi-implicit integration method is substantially better than the explicit-Euler algorithm used in Chapter 2, particularly for almost diagonal problems, the domains of dependence are identical. This can be seen by comparing Eqn. (3.19) to Eqn. (2.16). It is possible to construct semi-implicit integration methods that have larger domains of dependence than the Jacobi-semi-implicit integration method without requiring a matrix solution. In particular, there is the Seidel-semi-implicit method,

$$x(\tau_m) = x(\tau_{m-1}) + h_m [Dx(\tau_m) + (L + U)x(\tau_{m-1})]. \quad [3.23]$$

Solving for $x(\tau_m)$ leads to

$$x(\tau_m) = (I - h_m(L + D))^{-1}(I + h_m U)x(\tau_{m-1}). \quad [3.24]$$

where $(I - h_m(L + D))^{-1}$ is easy to compute because the matrix is triangular. The Seidel-semi-implicit method has stability properties that are similar to the Jacobi-semi-implicit method.

Theorem 3.6: If the matrix A in Eqn. (3.18) is diagonally dominant, with negative diagonal entries, or if A is lower or upper triangular, then the region of stability for the Seidel-semi-implicit method is the open left-half plane of ζ . ■

For the case of A diagonally dominant with negative diagonals, Theorem 3.6 follows from arguments similar to those used to prove Theorem 3.5. If A is lower triangular, the Seidel-semi-implicit algorithm is identical to implicit-Euler which is A-stable, and if A is upper triangular the algorithm is identical to the Jacobi-semi-implicit algorithm.

The Seidel-semi-implicit method does not have obviously better stability properties than the Jacobi-semi-implicit method, but it has the clear advantage of a larger domain of dependence. To see this, consider the expansion of $(I - h_m(L + D))^{-1}$ in Eqn. (3.24) for small h_m ,

$$x(\tau_m) = [I + h_m(L + D) + h_m^2(L + D)^2 + h_m^3(L + D)^3 + \dots][I + h_m U]x(\tau_{m-1}). \quad [3.25]$$

If A is lower triangular, the domain of dependence of the Seidel-semi-implicit method is exhaustive. As long as the lower triangular portion of A is nonzero, the domain of dependence of the Seidel-semi-implicit method will be larger than that of the Jacobi-semi-implicit method.

The Seidel-semi-implicit method includes the domain of dependence due to arbitrarily high powers of the lower triangular portion of A . The next semi-implicit method we will consider, the symmetric displacement algorithm[54,6], also includes the domain of dependence due to arbitrarily high powers of the upper-triangular portion of A . Applied to Eqn. (3.18), the symmetric displacement algorithm is the following two step process,

$$x(\tau_{m+1/2}) = x(\tau_m) + 0.25h_m[(2L + D)x(\tau_{m+1/2}) + (D + 2U)x(\tau_{m-1})]. \quad [3.26a]$$

$$x(\tau_m) = x(\tau_{m+1/2}) + 0.25h_m[(2U + D)x(\tau_{m+1/2}) + (D + 2L)x(\tau_m)]. \quad [3.26b]$$

Note that if A is diagonal, the symmetric displacement algorithm is precisely the trapezoidal rule.

The symmetric displacement algorithm has several important properties. The local truncation error is of order h^3 , unlike the other semi-implicit methods, whose error is of order h^2 [6]. In addition, it has the stability properties given in the following theorem.

Theorem 3.7: If the matrix A in Eqn. (3.18) is strictly diagonally dominant, with negative diagonal entries, or if A is symmetric, lower triangular or upper triangular, then the region of stability for the symmetric-displacement method is the open left-half plane of Φ . ■

The proof of Theorem 3.7 for the case where A strictly diagonally dominant with negative diagonal terms follows from the same reasoning as used in the proof of Theorem 3.5. The proof for case of A symmetric can be found in [6].

As indicated by Theorem 3.7, the stability properties of the symmetric displacement algorithm are better for near symmetric problems than those of the Seidel-semi-implicit method, but symmetric displacement has a smaller region of stability if the problem is almost lower triangular. The symmetric displacement algorithm is superior to the Seidel-Semi-implicit method in two important aspects, its local truncation is of a higher order, and it has a larger domain of dependence for problems that are not lower triangular. To show this, Eqn. (3.26a) and Eqn. (3.26b) are reorganized as

$$x(\tau_m) = [I - 0.25h_m(D + 2L)]^{-1} [I + 0.25h_m(D + 2U)] \quad [3.27]$$

$$[I - 0.25h_m(D + 2U)]^{-1} [I + 0.25h_m(D + 2L)] x(\tau_m).$$

The expansion of $[I - 0.25h_m(D + 2L)]^{-1}$ will include all the powers of L , and the expansion of $[I - 0.25h_m(D + 2U)]^{-1}$ will include all the powers of U . Note that this does not mean that the

symmetric displacement algorithm has an exhaustive domain of dependence. For example, the domain of dependence of A^2 is not necessarily the same as $L^2 + U^2$, there are possibly additional dependencies due to the cross-product terms LU and UL .

None of the semi-implicit methods mentioned above match the stiffly-stable implicit multistep method for either stability or domain of dependence. However, they have proved to be extremely useful for a variety of circuit simulation applications where either the problem is not that stiff, or is of a mostly diagonal or lower triangular form. For this reason, extensions of the semi-implicit methods mentioned above to the case where $C(x,u)$ is not diagonal have been pursued[55,6]. Similar results about region of stability for these extensions have been shown.

SECTION 3.4 - RELAXATION VS SEMI-IMPLICIT INTEGRATION

The relaxation-Newton algorithms described in Section 2 present a bound on the numerical integration timestep to insure that the relaxation converges. This bound is similar to the bound on the timestep to insure stability of the semi-implicit numerical integration methods. In order to demonstrate briefly the similarities of the two approaches, we will end this chapter by comparing the simplest of each type of method, the Jacobi-relaxation algorithm applied to solving the implicit-Euler equation, and the Jacobi-semi-implicit algorithm. Again, to keep the analyses simple, we will use the test problem of Eqn. (3.18)

The timepoint update equation for the Jacobi-semi-implicit algorithm is

$$x(\tau_m) = (I - h_m D)^{-1} [I + h_m(L + U)] x(\tau_{m-1}). \quad [3.27]$$

The iteration update equation of the Jacobi relaxation applied to implicit-Euler is

$$x^{k+1}(\tau_m) - x^k(\tau_m) = (I - h_m D)^{-1} [h_m(L + U)] [x^k(\tau_m) - x^{k-1}(\tau_m)]. \quad [3.28]$$

The semi-implicit method will be stable if

$$\rho[(I - h_m D)^{-1} (I + h_m (L + U))] < 1$$

and the relaxation will converge if

$$\rho[(I - h_m D)^{-1} (h_m (L + U))] < 1.$$

Both spectral radii will be less than 1 for any h_m if A is diagonally dominant and has negative diagonal elements. If A is not diagonally dominant, but has negative diagonal elements, the method that will allow the larger timestep will depend on the signs and magnitudes of the lower and upper triangular portions of A .

Although the size of the largest allowable timestep does not conclusively favor semi-implicit integration methods or relaxation methods, relaxation methods are clearly superior with respect to the relative domains of dependence. By carrying the relaxation iteration to convergence, it is assured that the information at a given timestep has propagated "far enough". Therefore, relaxation methods have the exhaustive domain of dependence property, and, as described above, the semi-implicit methods do not.

CHAPTER 4 - THE WAVEFORM RELAXATION ALGORITHM

The multistep numerical integration algorithms for solving ODE systems can become inefficient for large systems where different state variables are changing at very different rates. This is because the direct application of the integration method forces every differential equation in the system to be discretized identically, and this discretization must be fine enough so that the fastest changing state variable in the system is accurately represented. If it were possible to pick different discretization points, or timesteps, for each differential equation in the system so that each could use the largest timestep that would accurately reflect the behavior of its associated state variable, then the efficiency of the simulation would be greatly improved. This is referred to as the multirate problem[1], and numerical integration methods that allow for different state variables to use different timesteps are called multirate integration methods.

The selective trace technique for improving the efficiency of relaxation-Newton methods (Section 3.2) can be thought of as a limited multirate integration method. If, at a given timestep, the x_i variable is at its equilibrium (or stationary) point, and the x_j variables on which x_i depend do not change, then x_i will retain the value it had before the timestep. In fact, x_i will never be recomputed until some x_j on which it depends changes. If x_i is bypassed for several timesteps the effect is the same as if a large timestep were used to compute x_i . Therefore, selective trace algorithm exploits the kind of multirate behavior that stems from a system in which most of the variables remain at an equilibrium state. The selective trace algorithm can *not*, however, exploit a system for which the state variables have different rates of motion, but are not at equilibrium.

Techniques based on semi-implicit integration algorithms have been used both to achieve the kind of limited multirate integration described above, and to achieve full multi-rate integration methods[4,57]. However, as pointed out in Section 3.3, the semi-implicit integration algorithms do not have all of the properties that make a numerical method for circuit simulation robust. A different approach is to somehow decompose the differential equations *before* introducing discrete approxi-

mations. If the differential equations are solved independently, the numerical integration method used for each system can pick its own timestep, thereby achieving full multi-rate integration. In addition, since *any* numerical integration algorithm can be used to solve the decomposed systems, one that retains all the desirable numerical properties described in Chapter 2 can be used.

One method for decomposing differential equations is the family of Waveform Relaxation algorithms [11]. WR algorithms have captured considerable attention due to their favorable numerical properties and to the success in applying the WR algorithms to the solution of Metal-Oxide-Semiconductor (MOS) digital circuits. In this chapter the theoretical basis for the WR algorithm will be presented. Waveform relaxation will be introduced with a simple example, which will be followed by the general algorithm applied to systems of the form of Eqn. (2.2). Then a new proof of the convergence, one that demonstrates that the WR algorithm is a contraction mapping in a particular norm, will be presented. Extensions to the basic algorithm that allow for modified iteration equations (including discrete approximations) will be presented and it will be shown that the convergence of such extensions follows *directly* from the proof that the WR algorithm is a contraction mapping. We will end this chapter by presenting a derivative of the WR algorithm, the waveform relaxation-Newton(WRN) algorithm, which is the extension to nonlinear differential equations of the relaxation-Newton algorithm presented in Section 3.2.

SECTION 4.1 - THE BASIC WR ALGORITHM

We will start this section with a simple illustrative example, and then present the general WR algorithm. Consider the first-order two-dimensional differential equation in: $x(t) \in \mathbb{R}^2$ on $t \in [0, T]$.

$$\dot{x}_1 = f_1(x_1, x_2, t) \quad x_1(0) = x_{10} \quad [4.1a]$$

$$\dot{x}_2 = f_2(x_1, x_2, t) \quad x_2(0) = x_{20} \quad [4.1b]$$

The basic idea of the waveform-relaxation algorithm is to fix the waveform $x_2: [0, T] \rightarrow \mathbb{R}$ and solve Eqn. (4.1a) as a one dimensional differential equation in $x_1(t)$. The solution thus obtained for $x_1(t)$ can be substituted into Eqn. (4.1b) which will then reduce to another first-order differential equation in one variable, $x_2(t)$. Eqn. (4.1a) is then re-solved using the new solution for $x_2(t)$ and the procedure is repeated.

Alternately, fix the waveform $x_2(t)$ in Eqn. (4.1a) and fix $x_1(t)$ in Eqn. (4.1b) and solve both one dimensional differential equations simultaneously. Use the solution obtained for x_2 in Eqn. (4.1b) and the solution obtained for x_1 in Eqn. (4.1a) and re-solve both equations.

In this fashion, iterative algorithms have been constructed. Either replaces the problem of solving a differential equation in two variables by one of solving a sequence of differential equations in one variable. As described above, these two waveform relaxation algorithms can be seen as the analogues of the Gauss-Seidel and the Gauss-Jacobi techniques for solving nonlinear algebraic equations. Here, however, the unknowns are waveforms (elements of a function space), rather than real variables. In this sense, the algorithms are techniques for time-domain decoupling of differential equations.

The WR algorithm for solving systems of the form of Eqn. (2.2):

Algorithm 4.1 (WR Gauss-Seidel Algorithm for solving Eqn. (2.2))

The superscript k denotes the iteration count, the subscript i denotes the component index of a vector and ϵ is a small positive number.

$k \leftarrow 0$

Guess waveform $x^0(t)$; $t \in [0, T]$ such that $x^0(0) = x_0$

for example, set $x^0(t) = x_0$, $t \in [0, T]$;

repeat {

$k \leftarrow k + 1$

foreach ($i \in \{ 1, \dots, n \} \}$) {

solve

$$\begin{aligned} & \sum_{j=1}^i C_{ij}(x_1^k, \dots, x_i^k, x_{i+1}^{k-1}, \dots, x_n^{k-1}, u) \dot{x}_j^k + \\ & \sum_{j=i+1}^n C_{ij}(x_1^k, \dots, x_i^k, x_{i+1}^{k-1}, \dots, x_n^{k-1}, u) \dot{x}_j^{k-1} - \\ & f_i(x_1^k, \dots, x_i^k, x_{i+1}^{k-1}, \dots, x_n^{k-1}, u) = 0 \end{aligned}$$

for ($x_i^k(t)$; $t \in [0, T]$), with the initial condition $x_i^k(0) = x_{i0}$.

}
 } until ($\|x^k - x^{k-1}\| \leq \varepsilon$)

that is, until the iteration converges.

■

Note that the differential equation in Algorithm 4.1 has only one unknown variable x_i^t . The variables $x_{i+1}^{t-1}, \dots, x_n^{t-1}$ are known from the previous iteration and the variables x_1^t, \dots, x_{i-1}^t have already been computed. Also, the Gauss-Jacobi version of the WR Algorithm for Eqn. (2.2) can be obtained from Algorithm 4.1 by replacing the foreach statement with the forall statement and adjusting the iteration indices.

SECTION 4.2 - CONVERGENCE PROOF FOR THE BASIC WR ALGORITHM

If the matrix $C(x,u)$ is diagonally dominant and Lipschitz continuous with respect to x for all u then both the Gauss-Seidel and the Gauss-Jacobi versions of Algorithm 4.1 are guaranteed to converge. In [12], it was shown that the WR algorithm converges when applied to Eqn. (2.2) if $C(x,u)$ is diagonally dominant and independent of x . As many systems that are modelled in the form of Eqn. (2.2) include a dependence of C on x , we will present a more general convergence proof that extends the original theorem to include these systems. In addition, we will prove the WR algorithm is a contraction in a simpler norm than the one used in the original theorem.

We will prove the theorem by first showing that if $C(x,u)$ is diagonally dominant, then there exists a bound on the \dot{x}^k 's generated by the WR algorithm that is independent of k . Using this bound, we will show that the assumption that $C(x,u)$ is Lipschitz continuous implies there exists a norm on \mathbb{R}^n such that for arbitrary positive integers j and k ,

$$\|\dot{x}^{k+1}(t) - \dot{x}^{j+1}(t)\| \leq \gamma \|\dot{x}^k(t) - \dot{x}^j(t)\| + l_1 \|\dot{x}^{k+1}(t) - \dot{x}^{j+1}(t)\| + l_2 \|\dot{x}^k(t) - \dot{x}^j(t)\|$$

for some $\gamma < 1$ and $l_1, l_2 < \infty$ for all $t \in [0, T]$. In the properly chosen norm $\|\bullet\|_b$ on $C([0, T], \mathbb{R}^n)$ the above equation implies that

$$\|\dot{x}^{k+1} - \dot{x}^{j+1}\|_b \leq \hat{\gamma} \|\dot{x}^k - \dot{x}^j\|_b$$

where $\hat{\gamma} < 1$ and therefore the sequence $\{\dot{x}^k\}$ converges by the contraction mapping theorem. As $x^k(0) = x_0$ for all k , $\{x^k\}$ converges as well.

Before formally proving this basic WR convergence theorem we will state the well-known contraction mapping theorem[35], and a few lemmas which will be used in the course of the proof.

The Contraction Mapping Theorem: Let Y be a Banach space and $F: Y \rightarrow Y$. If F is such that $\|F(y) - F(x)\| \leq \gamma \|y - x\|$ for all $x, y \in Y$, for some $\gamma \in [0, 1)$, then F has a unique fixed point \tilde{y} such that $F(\tilde{y}) = \tilde{y}$. Furthermore, for any initial guess $y^0 \in Y$ the sequence $\{y^k \in Y\}$ generated by the fixed point algorithm $y^k = F(y^{k-1})$ converges uniformly to \tilde{y} .

Lemma 4.1: If $C(x, u) \in \mathbb{R}^{n \times n}$ is diagonally dominant uniformly over all $x \in \mathbb{R}^n$, $u \in \mathbb{R}^r$ then given any collection of vectors $\{x^1, \dots, x^n\}$, $x^i \in \mathbb{R}^n$, and any $u \in \mathbb{R}^r$, the matrix $C^p(x^1, \dots, x^n, u) \in \mathbb{R}^{n \times n}$ defined by $C_{ij}^p(x^1, \dots, x^n, u) \equiv C_{ij}(x^i, u)$ is also diagonally dominant. In other words, let C^p be the matrix constructed by setting the i^{th} row of C^p equal to the i^{th} row of the given matrix $C(x^i, u)$. Then this new matrix is also diagonally dominant. ■

Lemma 4.1 follows directly from the definition of diagonal dominance.

Lemma 4.2: Let $C \in \mathbb{R}^{n \times n}$ be any strictly diagonally dominant matrix. Let L strictly lower triangular, U strictly upper triangular, and D diagonal, be such that $C = L + D + U$. Then $\|D^{-1}(L + U)\|_{\infty} < 1$ and $\|(D + L)^{-1}U\|_{\infty} < 1$. ■

Lemma 4.2 is a standard result in matrix theory[28].

Lemma 4.3: Let $x, y \in C([0, T], \mathbb{R}^n)$. If there exists some norm on \mathbb{R}^n such that

$$\|\dot{x}(t)\| \leq \gamma \|y(t)\| + l_1 \|x(t)\| + l_2 \|y(t)\| \quad [4.2]$$

for some positive numbers $l_1, l_2 < \infty$ and $\gamma < 1$ then there exists a norm $\|\bullet\|_b$ on $C([0, T], \mathbb{R}^n)$ such that

$$\|\dot{x}\|_b \leq \alpha \|\dot{y}\|_b + l_1 \|x(0)\| + l_2 \|y(0)\| \quad [4.3]$$

for some positive number $\alpha < 1$. ■ **Proof of Lemma 4.3:**

Substituting $\int_0^t \dot{x}(\tau) d\tau + x(0)$ for $x(t)$ in Eqn. (4.2) and performing an analogous substitution for $y(t)$, multiplying the entire equation by e^{-bt} , and moving the norms inside the integral yields:

$$e^{-bt} \|\dot{x}(t)\| \leq \gamma e^{-bt} \|\dot{y}(t)\| + l_1 e^{-bt} \int_0^t \|\dot{x}(\tau)\| d\tau + l_1 e^{-bt} \|x(0)\| + \quad [4.4]$$

$$l_2 e^{-bt} \int_0^t \|\dot{y}(\tau)\| d\tau + l_2 e^{-bt} \|y(0)\|.$$

Let $\|\bullet\|_b$ be defined by $\|f\|_b \equiv \max_{t \in [0, T]} e^{-bt} \|f(t)\|$. This is a norm on $C([0, T], \mathbb{R}^n)$ for any finite positive number $b > 0$ and is equivalent to the uniform norm on $C([0, T], \mathbb{R}^n)$. Then Eqn. (4.4) implies

$$\|\dot{x}\|_b \leq \gamma \|\dot{y}\|_b + \max_{t \in [0, T]} [l_1 e^{-bt} \int_0^t e^{b\tau} d\tau \|\dot{x}\|_b + l_1 e^{-bt} \|x(0)\| +$$

$$l_2 e^{-bt} \int_0^t e^{b\tau} d\tau \|\dot{y}\|_b + l_2 e^{-bt} \|y(0)\|]$$

And since $e^{-bt} \int_0^t e^{b\tau} d\tau \leq \frac{1}{b}$, then for $b > l_1$ we can write

$$\|\dot{x}\|_b \leq \frac{\gamma + l_2 b^{-1}}{1 - l_1 b^{-1}} \|\dot{y}\|_b + l_1 \|x(0)\| + l_2 \|y(0)\|. \quad [4.5]$$

In this case γ is less than 1, so there exists a finite B for which $\frac{(\gamma + l_2 B^{-1})}{1 - l_1 B^{-1}} = \alpha < 1$. Let the b in Eqn. (4.5) be set equal to this B to get

$$\|\dot{x}\|_B \leq \alpha \|\dot{y}\|_B + l_1 \|x(0)\| + l_2 \|y(0)\| \quad [4.6]$$

which completes the proof. ■

Now we prove the following WR convergence theorem for systems of equations of the form of Eqn (2.2).

Theorem 4.1: If, in addition to the assumptions of Eqn. (2.2), $C(x(t), u(t)) \in \mathbb{R}^{n \times n}$ is strictly diagonally dominant uniformly over all $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^r$ and Lipschitz continuous with respect to $x(t)$ for all $u(t)$, and $x^0(t)$ is differentiable, then the sequence of waveforms $\{x^k\}$ generated by the Gauss-Seidel or Gauss-Jacobi WR algorithm will converge uniformly to the solution of Eqn. (2.2) for all bounded intervals $[0, T]$. ■

Proof of Theorem 4.1:

We will present the proof only for the Gauss-Seidel WR algorithm, as the proof for the Gauss-Jacobi case is almost identical. The equations for one iteration of the Gauss-Seidel WR algorithm applied to Eqn. (2.2) can be written in matrix form as

$$\hat{C}(x^{k+1}, x^k, u) \dot{x}^{k+1} = \hat{f}(x^{k+1}, x^k, u)$$

where $\hat{C}_{ij}(x^{k+1}, x^k, u) = C_{ij}(x_1^{k+1}, \dots, x_i^{k+1}, x_{i+1}^k, \dots, x_n^k, u)$ and $\hat{f}_i(x^{k+1}, x^k, u) = f_i(x_1^{k+1}, \dots, x_i^{k+1}, x_{i+1}^k, \dots, x_n^k, u)$. Let $\hat{C}(x^{k+1}, x^k, u) = L_{k+1} + D_{k+1} - U_{k+1}$ where L_{k+1} is strictly lower triangular, U_{k+1} is upper triangular, and D_{k+1} is diagonal (Note that by Lemma 4.1, the matrix \hat{C} is diagonally dominant because C is diagonally dominant). Rearranging the iteration equation yields:

$$\dot{x}^{k+1} = (L_{k+1} + D_{k+1})^{-1} [U_{k+1} \dot{x}^k + \hat{f}(x^{k+1}, x^k, u)]. \quad [4.7]$$

Taking the difference between Eqn. (4.7) at iteration $k + 1$ and at iteration $j + 1$ yields

$$\dot{x}^{k+1} - \dot{x}^{j+1} = (L_{k+1} + D_{k+1})^{-1} U_{k+1} \dot{x}^k - (L_{j+1} + D_{j+1})^{-1} U_{j+1} \dot{x}^j + \quad [4.8]$$

$$(L_{k+1} + D_{k+1})^{-1} \hat{f}(x^{k+1}, x^k, u) - (L_{j+1} + D_{j+1})^{-1} \hat{f}(x^{j+1}, x^j, u)$$

Using the Lipschitz continuity of \hat{f} and that $\|(L_{k+1} + D_{k+1})^{-1}\| < K$ for some $K < \infty$ independent of x and k (because $C(x, u)$ is uniformly diagonally dominant with respect to x) in Eqn. (4.8) leads to

$$\|\dot{x}^{k+1}(t) - \dot{x}^{j+1}(t)\| \leq l_1 K \|\dot{x}^{k+1}(t) - \dot{x}^{j+1}(t)\| + l_2 K \|\dot{x}^k(t) - \dot{x}^j(t)\| + \quad [4.9]$$

$$\|(L_{k+1} + D_{k+1})^{-1} - (L_{j+1} + D_{j+1})^{-1}\| \|\hat{f}(x^{j+1}, x^j, u)\| +$$

$$\|(L_{k+1} + D_{k+1})^{-1} U_{k+1} \dot{x}^k(t) - (L_{j+1} + D_{j+1})^{-1} U_{j+1} \dot{x}^j(t)\|$$

where l_1 is the Lipschitz constant of \hat{f} with respect to its first argument, and l_2 is the Lipschitz constant of \hat{f} with respect to its second argument. That $C(x, u)$ is uniformly diagonally dominant and Lipschitz continuous with respect to x for all u implies $(L_k + D_k)^{-1}$ and $(L_k + D_k)^{-1} U_k$ are also Lipschitz continuous in the same manner. It then follows that there exist some positive finite numbers k_1, k_2, k_3, k_4 such that

$$\|\dot{x}^{k+1}(t) - \dot{x}^{j+1}(t)\| = l_1 K \|\dot{x}^{k+1}(t) - \dot{x}^{j+1}(t)\| + l_2 K \|\dot{x}^k(t) - \dot{x}^j(t)\| + \quad [4.10]$$

$$[k_3 \|\dot{x}^{k+1}(t) - \dot{x}^{j+1}(t)\| + k_4 \|\dot{x}^k(t) - \dot{x}^j(t)\|] \|\hat{f}(x^{j+1}, x^j, u)\| +$$

$$[k_1 \|\dot{x}^{k+1}(t) - \dot{x}^{j+1}(t)\| + k_2 \|\dot{x}^k(t) - \dot{x}^j(t)\|] \|\dot{x}^k(t)\| + \gamma \|\dot{x}^k(t) - \dot{x}^j(t)\|$$

where k_1 is the Lipschitz constant of $(L_k + D_k)^{-1} U_k$ with respect to its first x argument (see definition of L_k, U_k and D_k above), k_2 is the Lipschitz constant with respect to the second x argument, k_3 and

k_4 are the Lipschitz constants for $(L_k + D_k)^{-1}$ with respect to its first and second x arguments, and γ is such that $\|(L_k + D_k)^{-1} U_k\| < \gamma < 1$ independent of k (by Lemma 4.2).

To establish a bound on the terms in Eqn. (4.10) involving $\|\dot{x}^k(t)\|$ and $\|\hat{f}(x^{k+1}, x^k, u)\|$ it is necessary to show that the \dot{x}^k 's and therefore the x^k 's and $\hat{f}(\bullet)$'s are bounded *a priori*. We prove such a bound exists in the following lemma.

Lemma 4.4: If $C(x, u)$ in Eqn. (2.2) is strictly diagonally dominant and Lipschitz continuous then the $\dot{x}^k(t)$'s produced by Algorithm 4.1 are bounded independent of k . ■

Proof of Lemma 4.4

If $\|\bullet\|$ is the l_∞ norm on \mathbb{R}^n , by Lemma 4.1 $\|(L_{k+1} + D_{k+1})^{-1} U_{k+1}\| < 1$. From Eqn. (4.7),

$$\|\dot{x}^{k+1}(t)\| < \gamma \|\dot{x}^k(t)\| + \|(L_{k+1} + D_{k+1})^{-1}\| \|\hat{f}(x^{k+1}(t), x^k(t), u)\| \quad [4.11]$$

for some positive number $\gamma < 1$. As $f(x, u)$ is globally Lipschitz continuous with respect to x , there exist finite positive constants l_1, l_2 such that

$$\|\hat{f}(x, y, u) - \hat{f}(w, z, u)\| < l_1 \|x - w\| + l_2 \|y - z\| \quad [4.12]$$

for all $u, x, y, w, z \in \mathbb{R}^n$. From Eqn. (4.11) and Eqn. (4.12) and using the fact that $\|(L_{k+1} + D_{k+1})^{-1}\|$ is bounded by some $K < \infty$ for all k :

$$\|\dot{x}^{k+1}(t)\| \leq \gamma \|\dot{x}^k(t)\| + l_1 K \|\dot{x}^{k+1}(t)\| + l_2 K \|\dot{x}^k(t)\| + K \|\hat{f}(0, 0, u)\| \quad [4.13]$$

Eqn. (4.13) is in the form to apply a slightly modified Lemma 4.3. Therefore there exists some $\|\bullet\|_b$ such that

$$\|\dot{x}^{k+1}\|_b \leq \alpha \|\dot{x}^k\|_b + (l_1 K + l_2 K) \|x(0)\| + K \|\hat{f}(0, 0, u)\| \quad [4.14]$$

where $\alpha < 1$. This implies that

$$\|\dot{x}^{k+1}\|_b \leq \frac{1}{1-\alpha}[(l_1K + l_2K)\|x(0)\| + K\|\hat{f}(0, 0, u)\|] + (\alpha)^k \|\dot{x}^0\|_b \quad [4.15]$$

for all k . Then, since $\|\dot{x}^0\|_b$ is bounded by assumption, and $\|\dot{x}^{k+1}\|_b = \max_{t \in [0, T]} e^{-bt} \|\dot{x}^{k+1}(t)\|$,

$$\|\dot{x}^{k+1}(t)\| < e^{bT} \left[\frac{1}{1-\alpha}[(l_1K + l_2K)\|x(0)\| + K\|\hat{f}(0, 0, u)\|] + \|\dot{x}^0\|_b \right] = \tilde{M} \quad [4.16]$$

which proves the lemma. ■

In Lemma 4.4 it was proved that $\|\dot{x}^k(t)\|$ is bounded *a priori* by some \tilde{M} . This implies $x^k(t)$ is bounded on $[0, T]$. Using the Lipschitz continuity property of \hat{f} , a bound, \tilde{N} , can be derived for $\|\hat{f}(x^{k+1}(t), x^k(t), u)\|$. Applying these bounds to Eqn. (4.10) we get

$$\|\dot{x}^{k+1}(t) - \dot{x}^{j+1}(t)\| \leq \gamma \|\dot{x}^k(t) - \dot{x}^j(t)\| + \quad [4.17]$$

$$(l_1K + k_1\tilde{M} + k_3\tilde{N}) \|x^{k+1}(t) - x^{j+1}(t)\| + (l_2K + \tilde{M}k_2 + k_4\tilde{N}) \|x^k(t) - x^j(t)\|$$

where $\gamma < 1$. Eqn. (4.17) is of the form to apply Lemma 4.3. As $x^{k+1}(0) - x^{j+1}(0) = 0$ for all k, j , Lemma 4.3 implies

$$\|\dot{x}^{k+1} - \dot{x}^{j+1}\|_b \leq \alpha \|\dot{x}^k - \dot{x}^j\|_b \quad [4.18]$$

for some norm on $C([0, T], \mathbb{R}^n)$ and for some $\alpha < 1$. As $C([0, T], \mathbb{R}^n)$ is complete in any one of the B -norms, by the contraction mapping theorem \dot{x}^k converges to some $\dot{x} \in C([0, T], \mathbb{R}^n)$ which is a fixed point of Eqn. (4.7). Any fixed point \dot{x} of Eqn. (4.7) is a solution to Eqn. (2.2) if $x(0) = x_0$, $x^k(0) = x_0$ for all k , therefore \dot{x}^k converges to the unique solution of Eqn. (2.2). The sequence $\{x^k\}$ converges because integration from 0 to T , which maps $\dot{x}(t)$ to $x(t)$, is a bounded continuous function.

■

SECTION 4.3 - NONSTATIONARY WR ALGORITHMS

Algorithm 4.1 is stationary in the sense that the equations that define the iteration process do not change with the iterations. A straight-forward generalization is to allow these iteration equations to change, and to consider under what conditions the relaxation still converges [13]. There are two major reasons for studying nonstationary algorithms. The solution of the ordinary differential equations in the inner loop of Algorithm 4.1 cannot be obtained exactly. Instead numerical methods compute the solution with some error which is in general controlled, but which cannot be eliminated. However, the discrete approximation can be interpreted as the exact solution to a perturbed system. Since the approximation changes with the solutions, the perturbed system changes with each iteration. Hence, practical implementations of WR that must compute the solution to the iteration equations approximately can be interpreted as nonstationary methods.

The second reason for studying nonstationary methods is that they can be used to improve the computational efficiency of the basic WR algorithm. An approach would be to improve the accuracy of the computation of the iteration equations as the relaxation approaches convergence. In this way, accurate solutions to the original system would still be obtained, but unnecessarily accurate computation of the early iteration waveforms, which are usually far from the final solution, is avoided.

In this section we show that nonstationary WR algorithms converge as a direct consequence of the contraction mapping property of the original WR algorithm. That is, given mild assumptions about the relationship between a general stationary contraction map and a nonstationary map, the nonstationary map will produce a sequence that will converge to within some tolerance. And if in the limit as $k \rightarrow \infty$ the nonstationary map approaches the stationary map, then the sequence generated by the nonstationary map will converge to the fixed point of the original map. In later sections we will lean on these results to guarantee the convergence of implementations of WR-based algorithms.

Theorem 4.2: Let Y be a Banach space and $F, F^k: Y \rightarrow Y$. Define $y^{k+1} = F(y^k)$ and $\tilde{y}^{k+1} = F^k(\tilde{y}^k)$. If F is a contraction mapping with contraction factor γ (See section 4.2), $\|F(y) - F^k(y)\| \leq \delta^k$ for

all $y \in Y$, and $z \in Y$, is such that $z = F(z)$, then for any $\varepsilon > 0$ there exists a $\delta < 1$ such that if $\delta^k < \delta$ for all k then $\lim_{k \rightarrow \infty} \|\tilde{y}^k - \tilde{y}^{k-1}\| < \varepsilon$ and $\lim_{k \rightarrow \infty} \|z - \tilde{y}^k\| < \frac{\delta}{1-\gamma}$. Furthermore, if $\lim_{k \rightarrow \infty} \delta^k \rightarrow 0$ then $\lim_{k \rightarrow \infty} \|\tilde{y}^k - \tilde{y}^{k-1}\| \rightarrow 0$ and $\lim_{k \rightarrow \infty} \|z - \tilde{y}^k\| \rightarrow 0$. ■

Proof of Theorem 4.2

Taking the norm of the difference between the k^{th} and $k+1^{\text{st}}$ iteration of the nonstationary algorithm we get:

$$\|\tilde{y}^{k+1} - \tilde{y}^k\| \leq \|F^{k+1}(\tilde{y}^k) - F^k(\tilde{y}^{k-1})\| \quad [4.19]$$

Given that $\|F^k(y) - F(y)\| < \delta^k$ for all $y \in Y$

$$\|\tilde{y}^{k+1} - \tilde{y}^k\| \leq \|F(\tilde{y}^k) - F(\tilde{y}^{k-1})\| + \delta^k + \delta^{k+1}. \quad [4.20]$$

Using the contraction property of F ,

$$\|\tilde{y}^{k+1} - \tilde{y}^k\| < \gamma \|\tilde{y}^k - \tilde{y}^{k-1}\| + \delta^k + \delta^{k+1}. \quad [4.21]$$

Unfolding the iteration equation into direct sum form,

$$\|\tilde{y}^{k+1} - \tilde{y}^k\|_b \leq \delta^{k+1} + \delta^k + \sum_{i=1}^k \gamma^{k-i} (\delta^i + \delta^{i-1}). \quad [4.22]$$

If $\delta^k < \delta$ for all k then from Eqn. (4.22)

$$\lim_{k \rightarrow \infty} \|\tilde{y}^{k+1} - \tilde{y}^k\| \leq 2\delta(1 + \frac{1}{1-\gamma}). \quad [4.23]$$

As $\gamma < 1$, $\lim_{k \rightarrow \infty} \|\tilde{y}^{k+1} - \tilde{y}^k\|$ can be made as small as desired by reducing δ , which proves the first part of Theorem 4.2.

Let y be the fixed point of F . The difference between the computed and the exact solution at the $k + 1^{\text{th}}$ iteration is

$$\|\tilde{y}^{k+1} - y\| = \|F^k(\tilde{y}^k) - F(y)\|. \quad [4.24]$$

Again using the contractive property of F and that $\|F(y) - F^k(y)\| \leq \delta^k$,

$$\|\tilde{y}^{k+1} - y\| = \gamma \|\tilde{y}^k - y\| + \delta^k. \quad [4.25]$$

Summing and taking the limit,

$$\lim_{k \rightarrow \infty} \|\tilde{y}^{k+1} - y\|_b \leq \frac{\delta}{1 - \gamma}, \quad [4.26]$$

which completes the proof of the first statement of Theorem 4.2. The second statement of the theorem follows from almost identical arguments. ■

In Section 4.2 we proved the WR iteration was a contraction mapping in the appropriate norm $\|\cdot\|_b$ on $C([0, T], \mathbb{R}^n)$ where B depended on the problem. To repeat the result from that section, it was shown that:

$$\|\tilde{x}^{k+1} - \tilde{x}^{j+1}\|_b \leq \alpha \|\tilde{x}^k - \tilde{x}^j\|_b$$

where $\alpha < 1$. This WR convergence result and Theorem 4.2 imply that using any "reasonable" approximation method to solve the WR iteration equations will still converge, *provided* the errors in the approximation are driven to zero. In addition, Theorem 4.2 indicates that it will be difficult to determine *a priori* how accurately the iteration equations must be solved to guarantee convergence to within a given tolerance, because an estimate of the contraction factor of the WR algorithm is required.

From Theorem 4.1, the WR is a contraction mapping with respect to $\dot{x}(t)$ in a B norm, Theorem 4.2 then implies that the WR iteration equations must be solved accurately with respect to $\dot{x}(t)$ in this B norm if the iterations are to converge. There is a more cumbersome proof of the WR convergence theorem in which it is shown that the WR algorithm is a contraction in $x(t)$, but in a larger B norm than the one used in the proof of Theorem 4.1, and the size of this B is a function of the magnitude of the off-diagonal terms of $C(x,u)$. With such a result, Theorem 4.2 implies that it is only necessary to control errors in the computation of $x(t)$ to guarantee iteration convergence. However, convergence in a larger B norm is in some sense a weaker type of convergence. So, in the case where $C(x,u)$ has non-zero off-diagonal terms, it is expected that more rapid convergence would be achieved if the $x^k(t)$'s are computed in a way that also guarantees that the $\dot{x}^k(t)$'s are globally accurate.

SECTION 4.4 - WAVEFORM RELAXATION-NEWTON METHODS

The WR algorithm is an extension to function spaces of the relaxation methods used to solve linear and nonlinear systems. It is also possible to extend the Newton-Raphson algorithm, and its function space extension also has practical applications. In particular, it is possible to approximately solve the WR iteration equations with one iteration of the Waveform-Newton algorithm, and this is the function space extension of the relaxation-Newton methods described in Section 3.2. In this section we will derive the function-space Newton method applied to systems of the form of Eqn. (2.2) and prove that the method has *global* convergence properties. We will then apply this method in conjunction with the WR algorithm to generate the Waveform-Relaxation-Newton (WRN) algorithm.

In order to derive a function-space extension to the Newton-Raphson algorithm, let $F(x)$ (from Eqn. (2.2)) be defined by

$$F(x) = C(x, u)\dot{x} - f(x, u) = 0 \quad x(0) = x_0 \quad [4.27]$$

where $x:[0,T] \rightarrow \mathbb{R}^n$, $u:[0,T] \rightarrow \mathbb{R}^r$ and is piecewise continuous; $C: \mathbb{R}^n \times \mathbb{R}^r \rightarrow \mathbb{R}^{n \times n}$ is such that $C(x, u)^{-1}$ exists and is uniformly bounded with respect to x, u ; and $f: \mathbb{R}^n \times \mathbb{R}^r \rightarrow \mathbb{R}^n$ is globally Lipschitz continuous with respect to x for all u . Applying the Newton-Raphson algorithm to find an x such that $F(x) = 0$ given some initial guess x^0 we get

$$x^{k+1} = x^k - J_F^{-1}(x^k)F(x^k) \quad [4.28]$$

where $J_F(x)$ is the Frechet derivative of $F(x)$ with respect to x . Note that in this case $J_F(x)$ is a matrix-valued function on $[0,T]$. That is, $J_F(x)$ is a matrix of waveforms.

Using the definition of the Frechet derivative, we can compute $J_F(x)$,

$$\lim_{\|h\| \rightarrow 0} (1/\|h\|) \|F(x+h) - F(x) - J_F(x)(h)\| \rightarrow 0. \quad [4.29]$$

Evaluating this limit for the $F(x)$ given in Eqn. (4.27) we get

$$F(x+h) - F(x) = C(x+h, u)(\dot{x} + \dot{h}) - C(x, u)\dot{x} - f(x+h, u) + f(x, u) \quad [4.30]$$

and approximating to order $\|h\|^2$

$$F(x+h) - F(x) = C(x, u)\dot{h} + \frac{\partial C(x, u)}{\partial x} h \dot{x} - \frac{\partial f(x, u)}{\partial x} h + O(\|h\|^2) \quad [4.31]$$

As Eqn. (4.29) applies only in the limit as $h \rightarrow 0$, Eqn. (4.31) implies

$$J_F(x)h = C(x, u)\dot{h} + \frac{\partial C(x, u)}{\partial x} h \dot{x} - \frac{\partial f(x, u)}{\partial x} h \quad [4.32]$$

Substituting the computed derivative into Eqn. (4.28) and rearranging we get

$$C(x^k, u)\dot{x}^{k+1} + \frac{\partial C(x^k, u)}{\partial x} (x^{k+1} - x^k)\dot{x}^k = f(x^k, u) + \frac{\partial f(x^k, u)}{\partial x} (x^{k+1} - x^k) \quad [4.33]$$

$$x^{k+1}(0) = x_0$$

We will refer to Eqn. (4.33) as the Waveform-Newton(WN) algorithm for solving Eqn. (2.2). It is, however, just the function-space extension of the classical Newton-Raphson algorithm.

Newton algorithms converge quadratically when the iterated value is close to the correct solution, but they do not, in general, have global convergence properties. The WN algorithm, along with inheriting the locally quadratic convergence properties of general Newton methods, will also converge globally, given mild assumptions on the behavior of $\frac{\partial C(x,u)}{\partial x}$ stated in the following theorem:

Theorem 4.3: For any system of the form of Eqn. (2.2) in which $\frac{\partial C(x,u)}{\partial x}$ is Lipschitz continuous with respect to x for all u and f is continuously differentiable, the sequence $\{x^k\}$ generated by the WN algorithm converges uniformly to the solution of Eqn. (2.2). ■

Proof of Theorem 4.3

For this proof of the convergence of the Waveform-Newton method we will assume that $C(x,u)$ is independent of x and u . as the proof for the general case is much more involved, and does not provide much further insight into the nature of the convergence. For the case $C(x,u) = C$ Eqn. (4.33) can be simplified to

$$\dot{x}^{k+1} = C^{-1}\dot{x}^k + C^{-1}f(x^k, u) + C^{-1} \frac{\partial f(x^k, u)}{\partial x} (x^{k+1} - x^k). \quad [4.34]$$

Taking the difference between Eqn. (4.34) at iteration $k + 1$ and the exact solution and substituting $(x^{k+1} - x) + (x - x^k)$ for $x^{k+1} - x^k$ yields

$$\dot{x}^{k+1} - \dot{x} = C^{-1} [f(x^k, u) - f(x, u)] + C^{-1} \left[\frac{\partial f(x^k, u)}{\partial x} ((x^{k+1} - x) + (x - x^k)) \right]. \quad [4.35]$$

As C has a bounded inverse by the assumptions following Eqn. (2.2), and that f is continuously differentiable on $[0, T]$ and Lipschitz continuous, $C^{-1} \frac{\partial f(x, u)}{\partial x}$ is bounded by some constant l_1 . With this bound,

$$\|\dot{x}^{k+1} - \dot{x}\| < l_1 \|x^k - x\| + l_1 \|x^{k+1} - x\| + l_1 \|x^k - x\|. \quad [4.36]$$

Lemma 4.3 can be applied to Eqn. (4.36) (with $\gamma = 0$). Therefore there exists some $b < \infty$ and $\alpha < 1$ such that

$$\|\dot{x}^{k+1} - \dot{x}\|_b \leq \alpha \|\dot{x}^k - \dot{x}\|_b. \quad [4.37]$$

Therefore $\{\dot{x}^k\}$ converges to \dot{x} , the fixed point of Eqn. (4.34). Given $x^k(0) = x_0$ for all k , $\{x^k\}$ converges to the solution of Eqn. (2.2) on any bounded interval. ■

As mentioned in the introduction it is possible to combine the Waveform-Newton method derived above with the WR algorithm to construct the waveform extension of the relaxation-Newton algorithms presented in Section 3.2[19]. The WR iteration equations are solved approximately by performing one step of this Newton method with each waveform relaxation iteration, to yield the following Waveform-Relaxation-Newton algorithm (WRN).

Algorithm 4.2 - (WRN Gauss-Seidel Algorithm for solving Eqn. (2.2))

The superscript k denotes the iteration count, the subscript $i \in \{1, \dots, N\}$ denotes the component index of a vector and ε is a small positive number.

$k \leftarrow 0$;

guess waveform $x^0(t)$; $t \in [0, T]$ such that $x^0(0) = x_0$

(for example, set $x^0(t) = x_0$, $t \in [0, T]$);

repeat {

$k \leftarrow k + 1$

for all (i in N) {

solve

$$\sum_{j=1}^{i-1} C_{ij}(x_1^k, \dots, x_{i-1}^k, x_i^{k-1}, \dots, x_n^{k-1}, u) \dot{x}_j^k + \frac{\partial C_{ii}(x_1^k, \dots, x_{i-1}^k, x_i^{k-1}, \dots, x_n^{k-1}, u)}{\partial x_i} (x_i^k - x_i^{k-1}) \dot{x}_i^k +$$

$$\sum_{j=i+1}^n C_{ij}(x_1^k, \dots, x_{i-1}^k, x_i^{k-1}, \dots, x_n^{k-1}, u) \dot{x}_j^{k-1} -$$

$$f_i(x_1^k, \dots, x_{i-1}^k, x_i^{k-1}, \dots, x_n^{k-1}, u) =$$

$$\frac{\partial f_i(x_1^k, \dots, x_{i-1}^k, x_{i+1}^{k-1}, \dots, x_n^{k-1}, u)}{\partial x_i} (x_i^k - x_i^{k-1}) = 0$$

for ($x_i^k(t)$; $t \in [0, T]$), with the initial condition $x_i^k(0) = x_{i0}$.

}
 } until ($\|x^k - x^*$ $\| \leq \varepsilon$)
 ■ .

Like Algorithm 4.1, each equation has only one unknown variable x_i^k , but in this case, each of the nonlinear equations has been replaced by a simpler time-varying linear problem.

Given the global convergence properties of both the original WR and the WN algorithms, it is not surprising that the WRN algorithm has global convergence properties. We will state the convergence theorem, but will not present the proof because it is quite similar to the proof of the basic WR and WN convergence theorems.

Theorem 4.4: If, in addition to the assumptions of Theorem 4.1, $\frac{\partial C(x, u)}{\partial x}$ is Lipschitz continuous with respect to x for all u ; then the sequence $\{x^k\}$ generated by the Gauss-Seidel or Gauss-Jacobi WRN algorithm converges to the solution of Eqn. (2.2) on all bounded intervals $[0, T]$. ■

The linear time-varying systems generated by the WRN algorithm are easier to solve numerically than the nonlinear iteration equations of the basic WR algorithm. For example, if an implicit multistep integration method is used to solve such a system, the implicit algebraic equations the multistep method generates will be linear. In addition, linear time-varying systems can be solved with a variety of efficient numerical techniques other than the standard discretization methods, such as collocation[58] and spectral methods[22].

CHAPTER 5 - DISCRETIZED WR ALGORITHMS

To compute the iteration waveforms for the WR algorithm it is usually necessary to solve systems of nonlinear ordinary differential equations. If multistep integration formulas are used to solve for the iteration waveforms, the differential equations that describe the decomposed systems will not be solved exactly. Therefore, the convergence theorem presented in Section 4.2 does not guarantee the convergence of this discretized WR algorithm. However, the discretized WR algorithm is a nonstationary method, and the theorems presented in Section 4.3 apply, and guarantee WR convergence to the solution of the given system of ODE's when the *global* discretization error is driven to zero with the WR iterations. Reducing the error with the iteration is also a reasonable practical approach to insuring the convergence of the WR algorithm under discretizations. Timesteps for numerical integration methods are usually chosen based on insuring that estimates of the local truncation error are kept below some supplied criteria. Reducing this criteria as relaxation iterations progress will insure that the WR algorithm will converge.

The view of the discretized WR algorithm as a nonstationary method, although simple and practical, lends no insight into why the discretized WR algorithm may not converge in some cases, and therefore provides no guidance for selecting a numerical integration method. It also does not allow for comparison to more classical integration methods. For this reason, in this chapter the interaction between WR algorithms and multistep integration methods will be considered in detail. In the first section, the discretized WR algorithm will be analyzed assuming that every differential equation in the system is discretized identically (hereafter referred to as the *global-timestep* case). A simple example will be presented that demonstrates a possible breakdown of the WR method under discretizations. The nonconvergence will be investigated by comparing the global-timestep discretized WR algorithm to the relaxation-Newton methods of Section 3.2. A strong comparison theorem for linear systems will be proved: the global timesteps required to insure WR convergence is identical to the timesteps required to insure convergence of the relaxation methods presented in

Section 3.2. A convergence theorem for the fixed global-timestep discretized WR algorithm will then be presented. In the second section, the global-timestep restriction will be lifted, and a theorem demonstrating the convergence of the multi-rate timestep case for systems in normal form will be presented.

SECTION 5.1 - THE GLOBAL TIMESTEP CASE

Consider the two-node inverter circuit in Fig. 5.1. The current equations at each node can be written by inspection, and are:

$$C\dot{x}_1 + g_1x_1 + g_2(x_1 - x_2) = 0 \quad [5.1]$$

$$C\dot{x}_2 + g_2(x_2 - x_1) + i_{m1}(x_1, x_2) + i_{m2}(x_1) = 0$$

$$x_1(0) = x_2(0) = 0.$$

In order to generate a simple linear example, i_{m1} , i_{m2} were linearized about the point where the input and output voltages were equal to half of the supply voltage. Time is normalized to seconds to obtain the following 2x2 example:

$$\dot{x}_1 = -x_1 + 0.1x_2 \quad [5.2]$$

$$\dot{x}_2 = -\lambda x_1 + -x_2$$

$$x_1(0) = x_2(0) = 0.$$

Note that the initial conditions given for the above example identify a stable equilibrium point.

The Gauss-Seidel WR iteration equations for the linear system example are:

$$\dot{x}_1^{k+1} = -x_1^{k+1} + 0.1x_2^k \quad [5.3]$$

$$\dot{x}_2^{k+1} = -\lambda x_1^{k+1} - x_2^{k+1}$$

$$x_1^{k+1}(0) = x_1^k(0) = x_2^{k+1}(0) = x_2^k(0) = 0.$$

Applying the Implicit-Euler numerical integration method with a fixed timestep h , ($\dot{x}(nh) = \frac{1}{h}[x(nh) - x((n-1)h)]$) to solve the decomposed equations yields the following recursion equation for $x_2^k(n)$:

$$x_2^{k+1}(n) = \frac{1}{1+h} x_2^{k+1}(n-1) - \frac{\lambda h}{(1+h)^2} \left[\frac{x_1(0)}{(1+h)^n} + 0.1h \sum_{j=1}^n (1+h)^{j-n} x_2^k(j) \right]. \quad [5.4]$$

For example, let $\lambda = 200$, $h = 0.5$ and as an initial guess use $x_2^0(nh) = nh$, which is far from the exact solution $x_2^0(nh) = 0$. The computed sequences for the initial guess and first, second and third iterations of Eqn. (5.4) are presented in Table 5.1.

TABLE 5.1 - IMPLICIT-EULER COMPUTED WR ITERATIONS					
STEP	TIME	INITIAL	ITER #1	ITER #2	ITER #3
0	0	0	0	0	0
1	0.5	0.5	-1.111	2.469	-5.487
2	1.0	1.0	-3.704	152	-32.92
3	5	5	-7.778	355	-111.6
4	2.0	2.0	-13.17	66.21	-281.3
5	2.5	2.5	-19.66	117.9	-587.5
6	3.0	3.0	-27.02	187.9	-1075
7	3.5	3.5	-35.07	276.0	-1786
8	4.0	4.0	-43.64	385	-2751
9	4.5	4.5	-52.60	502.9	-3992
10	5.0	5.0	-61.85	638.4	-5519

As the Table 5.1 indicates, the WR algorithm diverges for this example. In fact, Eqn. (5.4) indicates that the WR algorithm will converge only if

$$\frac{h}{(1+h)} < \frac{1}{\sqrt{0.1\lambda}} \quad [5.5]$$

The constraints on the timesteps for which the global-timestep discretized WR algorithm will converge is very similar to the constraints on the timesteps for which the relaxation-Newton algorithm applied to Eqn. (3.5) will converge (see Section 3.2). In fact, for linear problems there is the following comparison theorem.

Theorem 5.1: Let a consistent and stable multistep integration algorithm be applied to an arbitrary linear system of the form

$$C\dot{x}(t) = Ax(t) \quad x(0) = x_0 \quad [5.6]$$

where $C, A \in \mathbb{R}^{n \times n}$, C nonsingular, and $x(t) \in \mathbb{R}^n$. Assume further that the Gauss-Seidel(Jacobi) algebraic relaxation algorithm is used to solve the linear algebraic equations generated by the integration algorithm (as described in Section 3.2). Given a sequence of timesteps, $\{h_m\}$, the Gauss-Seidel(Jacobi) algebraic relaxation algorithm will converge at every step, for any initial guess, if and only if the global-timestep discretized Gauss-Seidel(Jacobi) WR algorithm, generated by solving the iteration equations with the same multistep integration algorithm and same timestep sequence, converges for any initial guess. ■

Proof of Theorem 5.1

The algebraic equations generated by applying a multistep integration algorithm to Eqn. (5.6) is

$$\sum_{i=0}^k \alpha_i C \hat{x}(\tau_{m-i}) = h_m \sum_{i=0}^l \beta_i A \hat{x}(\tau_{m-i}). \quad [5.7]$$

or reorganizing,

$$[C - h_m \beta_0 A] \hat{x}(\tau_m) + \sum_{i=1}^k \alpha_i C \hat{x}(\tau_{m-i}) - h_m \sum_{i=1}^l \beta_i A \hat{x}(\tau_{m-i}) = 0. \quad [5.8]$$

Let L_c, D_c, U_c be the strictly lower triangular, diagonal, and upper triangular portions of C . Similarly, let L_a, D_a, U_a be the strictly lower triangular, diagonal, and upper triangular portions of A . Using this notation, the Gauss-Seidel relaxation iteration equation applied to solving Eqn. (5.8) for $x(\tau_m)$ is

$$[(L_c + D_c) - h_m \beta_0 (L_a + D_a)] \hat{x}^k(\tau_m) + [U_c - h_m \beta_0 U_a] \hat{x}^{k-1}(\tau_m) +$$

$$\sum_{i=1}^k \alpha_i C \hat{x}(\tau_{m-i}) - h_m \sum_{i=1}^l \beta_i A \hat{x}(\tau_{m-i}) = 0.$$

Taking the difference between the k and $k-1$ iteration and substituting $\delta^k(\tau_m)$ for $x^k(\tau_m) - x^{k-1}(\tau_m)$ leads to

$$[(L_c + D_c) - h_m \beta_0 (L_a + D_a)] \delta^k(\tau_m) = -[U_c - h_m \beta_0 U_a] \delta^{k-1}(\tau_m) \quad [5.9]$$

from which it follows that the relaxation will converge at the m^{th} for any initial guess if and only if the spectral radius of

$$[(L_c + D_c) - h_m \beta_0 (L_a + D_a)]^{-1} [U_c - h_m \beta_0 U_a] \quad [5.10]$$

is less than one.

If the Gauss-Seidel WR algorithm is used to solve Eqn. (5.6), the iteration equation for $x(t)$ is (using the above notation),

$$(L_c + D_c) \dot{x}^{k+1}(t) + U_c \dot{x}^k(t) = (L_a + D_a) x^{k+1}(t) + U_a x^k(t). \quad [5.11]$$

Applying the multistep integration algorithm to solve Eqn. (5.11) for x^{k+1} yields

$$[(L_c + D_c) - h_m \beta_0 (L_a + D_a)] \hat{x}^k(\tau_m) + [U_c - h_m \beta_0 U_a] \hat{x}^{k-1}(\tau_m) + \quad [5.12]$$

$$\sum_{i=1}^k \alpha_i [(L_c + D_c) \hat{x}^k(\tau_{m-i}) + U_c \hat{x}^{k-1}(\tau_{m-i})] -$$

$$h_m \sum_{i=1}^l \beta_i [(L_a + D_a) \hat{x}^k(\tau_{m-i}) + U_a \hat{x}^{k-1}(\tau_{m-i})] = 0.$$

taking the difference between the k and $k - 1$ iteration leads to

$$[(L_c + D_c) - h_m \beta_0 (L_a + D_a)] \delta^k(\tau_m) + [U_c - h_m \beta_0 U_a] \delta^{k-1}(\tau_m) + \quad [5.13]$$

$$\sum_{i=1}^k \alpha_i [(L_c + D_c) \delta^k(\tau_{m-i}) + U_c \delta^{k-1}(\tau_{m-i})] -$$

$$h_m \sum_{i=1}^l \beta_i [(L_a + D_a) \delta^k(\tau_{m-i}) + U_a \delta^{k-1}(\tau_{m-i})] = 0.$$

To show that the discretized WR algorithm will only converge if the algebraic relaxation converges, let l be a timestep for which the spectral radius of the matrix in Eqn. (5.10) is not less than one. Use as an initial guess any sequence for which the first $l - 1$ points are the exact solution to the discretized problem. Then $\delta^k(\tau_m) = 0$ for $m < l$, and Eqn. (5.13) is again identical to Eqn. (5.9), and is not convergent.

An inductive argument is used to prove that if the algebraic relaxation is convergent then the discretized WR algorithm is convergent. Assume that the theorem holds for $m < l$ then $\delta^k(\tau_{l-1})$ will go to zero as $k \rightarrow \infty$. As this occurs, Eqn. (5.13) for the l^{th} step converges to Eqn. (5.9). The algebraic relaxation converges and therefore the spectral radius of the matrix in Eqn. (5.10) for the l^{th} step

is less than one. This implies that Eqn. (5.9) represents a contraction mapping in some norm at the l^{th} step, and the results of Section 4.3 can be applied to guarantee that Eqn. (5.13) converges at the l^{th} step. Note that $\delta^k(\tau_m) = 0$ for all $m \leq 0$, and therefore Eqn. (5.13) is identical to Eqn. (5.9) for $m = 1$ which completes the induction. ■

The above theorem holds for any system of the form of Eqn. (2.2) if it is assumed that an arbitrarily close initial guess for each of the relaxation schemes is available. Although this is not a realistic assumption, it does indicate that even for nonlinear systems the two algorithms present very similar timestep constraints for a numerical integration method.

SECTION 5.2 - GLOBAL FIXED-TIMESTEP WR CONVERGENCE THEOREM

It is possible to generalize the proof of Theorem 5.1 to a proof for the global-timestep discretized WR algorithm for nonlinear problems (but, as mentioned above, the comparison to the relaxation-Newton methods would no longer hold). A different approach will be taken, because the approach followed in Theorem 5.1 does not prove the the discretized WR algorithm converges on a fixed time interval as the timesteps become small.

To illustrate this point by example, consider solving Eqn. (5.3) using explicit-Euler. The recursion equation for the $x_2^k(n)$'s is:

$$x_2^{k+1}(n+1) = (1-h)x_2^{k+1}(n) - 0.1\lambda h^2[(1-h)^n x_1^k(0) + \sum_{j=1}^{n-1} (1-h)^{n-1-j} x_2^k(j)]$$

The computed sequences $\{x_2^{k+1}\}$'s for the initial guess and first, second and third iterations of the above equation are given in Table 5.2, for the case of $\lambda = 200$, $h = 0.5$ and $x_2^0(nh) = nh$.

As the table indicates, the explicit-Euler discretized WR algorithm converges in just the manner predicted by Theorem 5.1, a step (or two) with each iteration. In the same example, if half the timestep is used, similar results are achieved. That is, the relaxation converges two steps with each iteration. If it were the case that no matter how small the timesteps became, each relaxation iteration

TABLE 5.2 - EXPLICIT-EULER COMPUTED WR ITERATIONS					
STEP	TIME	INITIAL	ITER #1	ITER #2	ITER #3
0	0	0	0	0	0
1	0.5	0.5	0	0	0
2	1.0	1.0	0	0	0
3	5	5	-0.625	0	0
4	2.0	2.0	-1.875	0	0
5	2.5	2.5	3.594	0.7813	0
6	3.0	3.0	-5.625	3.125	0
7	3.5	3.5	-7.852	7.422	-0.977
8	4.0	4.0	-10.19	13.67	-4.883
9	4.5	4.5	-12.61	21.63	-13.92
10	5.0	5.0	-15.06	30.96	-29.79

resulted only in two more timesteps converging, then given a fixed interval of interest, the WR algorithm would *not* be convergent in the limit as the timesteps approached zero. This is not the case for this example, or in general for the discretized WR algorithm. If, for example, $h = 0.05$ then the relaxation converges in a more uniform manner, where the value at each timestep rapidly approaches its limit point.

In Section 4.2, the WR algorithm was shown to be a contraction mapping, specifically:

$$\max_{[0,T]} e^{-\beta t} \|\dot{x}^k(t) - \dot{x}^l(t)\| \leq \gamma \max_{[0,T]} e^{-\beta t} \|\dot{x}^{k-1}(t) - \dot{x}^{l-1}(t)\|$$

where $\gamma, \beta \in \mathbb{R}$ are dependent on the problem, and $\gamma < 1$. If T is chosen small enough, then $\gamma e^{\beta T} = \hat{\gamma} < 1$ and the norm becomes

$$\max_{[0,T]} \|\dot{x}^k(t) - \dot{x}^l(t)\| \leq \hat{\gamma} \max_{[0,T]} \|\dot{x}^{k-1}(t) - \dot{x}^{l-1}(t)\|.$$

That is, the WR algorithm converges uniformly over small time intervals (This point will be discussed further in Section 6.2). The next theorem will be an analogous proof for the discretized case. It will be shown that the fixed global timestep discretized WR algorithm is a contraction in a β norm (the technique was first applied to proving discrete WR convergence in [29]).

Formally, demonstrating that the discretized WR is a contraction in a β norm implies the convergence of the discretized WR algorithm because of the contraction mapping theorem. Intuitively, that the discretized WR algorithm converges in a β norm implies an underlying uniformity that guarantees convergence over a fixed time interval as the timesteps shrink to zero. This is the distinction between Theorem 5.1 and the next theorem.

Theorem 5.2: If, in addition to the assumptions of Theorem 4.1, f in Eqn. (2.2) is differentiable, and the WR iteration equations are solved using a stable, consistent, multistep integration method with a fixed timestep h , then the sequences $\{x^t(n)\}$ generated by the Gauss-Seidel or Gauss-Jacobi discretized WR algorithm will converge for all $h > 0$. ■

Before proving Theorem 5.2, some standard notation [1,59] will be presented that will also be used in the next section. The fixed-timestep multistep integration algorithms applied to

$$\dot{x}(t) = f(x(t)) \quad x(0) = x_0, \quad [5.14]$$

where $x: [0, T] \rightarrow \mathbb{R}^n$, $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$ can be represented by backward shift operators. That is, given

$$\sum_{i=0}^k \alpha_i \hat{x}(\tau_{m-i}) = h_m \sum_{i=0}^l \beta_i f(\hat{x}(\tau_{m-i})) \quad [5.15]$$

we can define

$$\rho(\hat{x}(\tau^m)) = \sum_{i=0}^k \alpha_i \hat{x}(\tau_{m-i}) \quad [5.16a]$$

and

$$\sigma(f(\hat{x}(\tau^m))) = \sum_{i=0}^l \beta_i f(\hat{x}(\tau_{m-i})) \quad [5.16b]$$

Eqn. (5.15) can then be written compactly as

$$\rho(\hat{x}(\tau_m)) = h_m \sigma(f(\hat{x}(\tau_{m-1}))) \quad [5.17]$$

If it is assumed that the operator ρ can be inverted, i.e. that $\hat{x}(\tau_m)$ can be expressed as a function of the right-hand side, then Eqn. (5.17) can be written in the form

$$\hat{x}(\tau_m) = h_m \rho^{-1} \sigma f(\hat{x}(\tau_m)). \quad [5.18]$$

When such an inverse of ρ exists, it can be shown that Eqn. (5.18) is equivalent to

$$\hat{x}(\tau_m) = \sum_{j=0}^m \gamma_j f(\hat{x}(\tau_{m-j})) + x(0). \quad [5.19]$$

As an example, consider implicit-Euler applied to Eqn. (5.14). The usual form for the discrete equations is,

$$\hat{x}(\tau_m) - \hat{x}(\tau_{m-1}) = f(\hat{x}(\tau_m)) \quad [5.20]$$

which is in the form of Eqn. (5.17). The implicit-Euler discrete equations can also be expressed in the form of Eqn. (5.19),

$$\hat{x}(\tau_m) = \sum_{j=0}^m f(\hat{x}(\tau_{m-j})) + x(0). \quad [5.21]$$

The solution to Eqn. (5.21) is obviously identical to the solution to Eqn. (5.20). The form of Eqn. (5.17) for the trapezoidal rule is

$$\hat{x}(\tau_m) - \hat{x}(\tau_{m-1}) = 0.5[f(\hat{x}(\tau_m)) + f(\hat{x}(\tau_{m-1}))], \quad [5.22]$$

which can also be expressed in the form of Eqn. (5.19) as

$$\hat{x}(\tau_m) = 0.5[f(\hat{x}(\tau_m)) + f(x(0))] + \sum_{j=1}^{m-1} f(\hat{x}(\tau_{m-j})) + x(0). \quad [5.23]$$

The following lemma, a special case of a theorem proof in [30], will be the key result used in the course of the proof of Theorem 5.2.

Lemma 5.1: Let $H(b)$ be the map that represents one iteration of the algebraic Gauss-Seidel or Gauss-Jacobi relaxation algorithm applied to an equation system of the form $f(x) - b = 0$, where $x, b \in \mathbb{R}^n, f: \mathbb{R}^n \rightarrow \mathbb{R}^n$. If f is such that the Jacobian of $f, \frac{\partial f}{\partial x}$, exists for all x , is strictly diagonally dominant uniformly over x , then $H(b)$ is a contraction mapping in the l_∞ norm and is a Lipschitz continuous function of b . ■

Proof of Lemma 5.1:

As usual, only the Gauss-Seidel case will be proved. It will be shown that if the Gauss-Seidel relaxation algorithm is used to solve $f(x) - b = 0$, then the map implicitly defined by one iteration of the relaxation, $H(b)$, is such that given $x^i, y^i \in \mathbb{R}^n$, two arbitrary points,

$$\|H(b)x^i - H(b)y^i\|_\infty \leq \gamma \|x^i - y^i\|_\infty \quad [5.24]$$

where $\gamma < 1$.

Define

$$x^{k,j} = (x_1^{k+1}, \dots, x_i^{k+1}, x_{i+1}^k, \dots, x_n^k)^T. \quad [5.25]$$

The iteration equation for x_i^{t+1} is implicitly defined by

$$f_1(x_1^{k+1}, x_2^k, \dots, x_n^k) - b_1 = 0 \quad [5.26a]$$

or, using the above notation $f_1(x^{k+1,1}) - b = 0$. In the same notation, the implicit iteration equation for y_1 is

$$f_1(y^{k+1,1}) - b = 0. \quad [5.26b]$$

Define the function $\psi(t) = f_1(t x^{k+1,1} + (1-t)y^{k+1,1}) - b$ where $t \in [0,1]$. Clearly, $\psi(0) = \psi(1) = 0$. By Rolle's theorem there exists a $t_0 \in (0,1)$ such that

$$\psi'(t_0) = 0 = \sum_{j=1}^n \frac{\partial f_1}{\partial x_j} (x^{k+1,1} + (1-t_0)y^{k+1,1})(x_j^{k+1,1} - y_j^{k+1,1}) \quad [5.27]$$

Reorganizing,

$$\frac{\partial f_1}{\partial x_1} (x^{k+1,1} + (1-t_0)y^{k+1,1})(x_1^{k+1,1} - y_1^{k+1,1}) = - \sum_{j=1}^n \frac{\partial f_1}{\partial x_j} (x^{k+1,1} + (1-t_0)y^{k+1,1})(x_j^{k+1,1} - y_j^{k+1,1}) \quad [5.28]$$

Dividing Eqn. (5.28) by $\frac{\partial f_1}{\partial x_1}$, which is bounded away from zero by the uniform strict diagonal dominance of $\frac{\partial f}{\partial x}$, and using the fact that $|x_j^k - y_j^k| \leq \|x^k - y^k\|_\infty$ by definition, we get

$$|x_1^{k+1} - y_1^{k+1}| = - \sum_{j=1}^n \left| \frac{\frac{\partial f_1}{\partial x_j} (x^{k+1,1} + (1-t_0)y^{k+1,1})}{\frac{\partial f_1}{\partial x_1} (x^{k+1,1} + (1-t_0)y^{k+1,1})} \right| \|x^k - y^k\|_\infty \quad [5.29]$$

Using the property of f that the Jacobian is strictly diagonally dominant uniformly in x leads to

$$|x_1^{k+1} - y_1^{k+1}| \leq \gamma_1 \|x^k - y^k\|_\infty$$

where $\gamma_1 < 1$. This proves that

$$|H_1(x^k) - H_1(y^k)| \leq \gamma_1 \|x^k - y^k\|_\infty. \quad [5.30]$$

A similar argument can be used to show

$$|H_i(x^k) - H_i(y^k)| \leq \gamma_i \|x^k - y^k\|_\infty.$$

where $\gamma_i < 1$. Then if γ is chosen to be the maximum of the γ_i 's, $\gamma < 1$ and

$$\|H(x^k) - H(y^k)\|_\infty \leq \gamma \|x^k - y^k\|_\infty. \quad [5.31]$$

which proves the first part of the theorem. (for a more detailed proof of the general cases, see [30]).

That H is a Lipschitz continuous function of b can be seen by examining the implicitly defined H_1 ,

$$f_1(x_1^{k+1}, x_2^k, \dots, x_n^k) - b_1 = 0$$

which is solved for x_1^{k+1} . A simple application of the implicit function theorem[35] implies that if $\frac{\partial f_1}{\partial x_1}$ is bounded away from zero uniformly in x , then x_1^{k+1} is a Lipschitz continuous function of b_1 . The argument can be carried inductively to show that for each i , $H_j, j \leq i$ is a Lipschitz continuous function of b and that therefore $H(b)$ is Lipschitz continuous with respect to b . ■

The formal definition of the β norm for a sequence is given below.

Definition 5.1: For a sequence generated by a fixed-timestep numerical integration algorithm, $\{x(\tau_m)\}$, the β norm of the sequence is defined as

$$\|\{x(\tau_m)\}\|_\beta = \max_m e^{-Bhm} \|x(\tau_m)\| \quad [5.32]$$

where h is the fixed timestep and $B \in \mathbb{R}$. ■

The following simple lemma will be useful for the proof of Theorem 5.2.

Lemma 5.2: Given an arbitrary sequence, $\{x(\tau_m)\}$, the following inequality holds,

$$\left\| \sum_{i=1}^m \gamma_i x(\tau_{m-i}) \right\|_B \leq M \frac{e^{-Bhm}}{1 - e^{-Bhm}} \left\| \{x(\tau_m)\} \right\|_B \quad [5.33]$$

where $M = \max_i |\gamma_i|$. ■

Proof of Lemma 5.2:

The proof of Lemma 5.2 follows from a simple algebraic argument. From Definition 5.1,

$$\left\| \sum_{i=1}^m \gamma_i x(\tau_{m-i}) \right\|_B = \max_m e^{-Bhm} \left\| \sum_{i=1}^m \gamma_i x(\tau_{m-i}) \right\| \quad [5.35]$$

Using the norm properties, the term

$$e^{-Bhm} \left\| \sum_{i=1}^m \gamma_i x(\tau_{m-i}) \right\| \quad [5.36]$$

can be bounded by

$$e^{-Bhm} \sum_{i=1}^m |\gamma_i| \left\| x^{\wedge^{k+1}}(\tau_{m-i}) \right\|. \quad [5.37]$$

Inserting $e^{B(m-i)h} e^{-B(m-i)h} = 1$ into Eqn. (5.37) yields

$$e^{-Bhm} \sum_{i=1}^m |\gamma_i| e^{Bh(m-i)} e^{-Bh(m-i)} \|x^{k+1}(\tau_{m-i})\| \quad [5.38]$$

As

$$e^{-Bh(m-i)} \|x^{k+1}(\tau_{m-i})\| \leq \|x^{k+1}(\tau_m)\|_B, \quad [5.39]$$

Eqn. (5.38) leads to

$$e^{-Bhm} \sum_{i=1}^m |\gamma_i| e^{Bh(m-i)} \|x^{k+1}(\tau_m)\|_B. \quad [5.40]$$

Reorganizing,

$$\left[\sum_{i=1}^m |\gamma_i| e^{-Bhi} \right] \|x^{k+1}(\tau_m)\|_B. \quad [5.41]$$

If $|\gamma_i|$ is bounded above by M , then Eqn. (5.41) is bounded by

$$M \left[\sum_{i=1}^m e^{-Bhi} \right] \|x^{k+1}(\tau_m)\|_B. \quad [5.42]$$

Given that e^{-Bhi} is always positive, the following inequality holds,

$$\sum_{i=1}^m e^{-Bhi} \leq \sum_{i=1}^{\infty} e^{-Bhi},$$

and from the infinite series summation formula

$$\sum_{i=1}^{\infty} e^{-Bhi} = \frac{e^{-Bh}}{1 - e^{-Bh}}.$$

Using the two in Eqn. (5.42) produces the conclusion of the lemma. ■

Proof of Theorem 5.2:

As before, only the Gauss-Seidel case will be proved. In order to insure charge conservation, the decomposed differential equations generated by the WR algorithm are solved using charge as the state variable. That is, the multistep integration algorithm is applied to

$$\dot{q}_i(x^{kj}(t), u(t)) = f_i(x^{kj}(t), u(t)) \quad [5.43]$$

where $x^{kj}(t)$, defined in Eqn. (5.25), is usually the vector of node voltages. A proof for the case where x is used as the state variable is given in [29]. Applying the multistep integration algorithm using the notation described above, and assuming $h_m = h$ for all m ,

$$\rho(q_i(\hat{x}^{kj}(\tau_m), u(\tau_m))) = h\sigma(f_i(\hat{x}^{kj}(\tau_m), u(\tau_m))). \quad [5.44]$$

Solving, using the "inverse" operator yields

$$q_i(\hat{x}^{kj}(\tau_m), u(\tau_m)) = h\rho^{-1}\sigma(f_i(\hat{x}^{kj}(\tau_m), u(\tau_m))). \quad [5.45]$$

Using the sum form for $\rho^{-1}\sigma$, and pulling out the leading term,

$$q_i(\hat{x}^{kj}(\tau_m), u(\tau_m)) = h\gamma_0(f_i(\hat{x}^{kj}(\tau_m), u(\tau_m))) - \quad [5.46]$$

$$h \sum_{j=1}^m \gamma_j (f_i(\hat{x}^{k,j}(\tau_{m-j}), u(\tau_{m-j}))) + q_i(x(0), u(0)) = 0$$

Define $F_i(\hat{x}(\tau_m))$ as

$$F_i(\hat{x}(\tau_m)) = q_i(\hat{x}(\tau_m), u(\tau_m)) - h\gamma_0 f_i(\hat{x}(\tau_m), u(\tau_m)) \quad [5.47]$$

and define $b(\hat{x}_{past}, k) \in \mathbb{R}^n$ by

$$b_i(\hat{x}_{past}, k) = h \sum_{j=1}^m \gamma_j (f_i(\hat{x}^{k,j}(\tau_{m-j}), u(\tau_{m-j}))) + q_i(x(0), u(0)) \quad [5.48]$$

where \hat{x}_{past}, k is used to denote the fact that b is a function of $\hat{x}^k(\tau_l)$ and $\hat{x}^{k-1}(\tau_l)$ for all $l < m$. Then Eqn. (5.45) is identical to one iteration of the algebraic Gauss-Seidel algorithm applied to solving

$$F(\hat{x}(\tau_m)) - b(\hat{x}_{past}, k) = 0. \quad [5.49]$$

for $\hat{x}(\tau_m)$. As in Lemma 5.2, \hat{x}^{k+1} can be written in terms of the map, $H(b(\hat{x}_{past}, k))$, defined implicitly by the Gauss-Seidel relaxation algorithm applied to Eqn. (5.49),

$$\hat{x}^{k+1}(\tau_m) = H(b(\hat{x}_{past}, k))\hat{x}^k(\tau_m). \quad [5.50]$$

To prove that the iteration described by Eqn. (5.50) is a contraction mapping on the sequence $\{\hat{x}^k(\tau_m)\}$, it will be shown that given two arbitrary sequences, $\{\hat{x}^k(\tau_m)\}$, and $\{\hat{y}^k(\tau_m)\}$,

$$\max_m e^{-Bmh} \|\hat{x}^{k+1}(\tau_m) - \hat{y}^{k+1}(\tau_m)\| \leq \max_m e^{-Bmh} \|\hat{x}^k(\tau_m) - \hat{y}^k(\tau_m)\| \quad [5.51]$$

where we will use the notation

$$\max_m e^{-Bmh} \|\hat{x}^k(\tau_m)\| = \|\{\hat{x}^k(\tau_m)\}\|_B \quad [5.52]$$

To start, Eqn. (5.50) leads to the following equation for the difference between the two sequences at the m^{th} step,

$$\hat{x}^{k+1}(\tau_m) - \hat{y}^{k+1}(\tau_m) = H(b(\hat{x}_{past}, k))\hat{x}^k(\tau_m) - H(b(\hat{y}_{past}, k))\hat{y}^k(\tau_m). \quad [5.53]$$

Breaking into separate differences,

$$\hat{x}^{k+1}(\tau_m) - \hat{y}^{k+1}(\tau_m) = H(b(\hat{x}_{past}, k))\hat{x}^k(\tau_m) - H(b(\hat{x}_{past}, k))\hat{y}^k(\tau_m) + \quad [5.54]$$

$$H(b(\hat{x}_{past}, k))\hat{y}^k(\tau_m) - H(b(\hat{y}_{past}, k))\hat{y}^k(\tau_m)$$

and taking l_∞ norms,

$$\|\hat{x}^{k+1}(\tau_m) - \hat{y}^{k+1}(\tau_m)\|_\infty \leq \|H(b(\hat{x}_{past}, k))\hat{x}^k(\tau_m) - H(b(\hat{x}_{past}, k))\hat{y}^k(\tau_m)\|_\infty + \quad [5.55]$$

$$\|H(b(\hat{x}_{past}, k))\hat{y}^k(\tau_m) - H(b(\hat{y}_{past}, k))\hat{y}^k(\tau_m)\|_\infty$$

At this point we will demonstrate that for small h , Eqn. (5.55) satisfies the assumptions of Lemma 5.1. It is assumed that the Jacobian of q with respect to x , $C(x(t), u(t))$, is strictly diagonally dominant uniformly in x . By definition, this assumption implies that there exists an $\varepsilon > 0$ such that

$$|C_{ii}(x(t), u(t))| > \varepsilon + \sum_{j \neq i} |C_{ij}(x(t), u(t))| \quad [5.56]$$

Let $l > 0$ be the Lipschitz constant of f with respect to x . Assuming f is differentiable, if $\gamma_0 = 0$ (and therefore the method is explicit) or if $h < \left| \frac{\varepsilon}{\gamma_0 l} \right|$, then $\frac{\partial F}{\partial \hat{x}}(\hat{x}(\tau_m))$ is strictly diagonally dominant.

Assuming h is small enough that $\frac{\partial F}{\partial \hat{x}}(\hat{x}(\tau_m))$ is strictly diagonally dominant, then Lemma 5.1 can be applied to show

$$\|H(b(\hat{x}_{past}, k))\hat{x}^k(\tau_m) - H(b(\hat{x}_{past}, k))\hat{y}^k(\tau_m)\|_{\infty} \leq \gamma \|\hat{x}^k(\tau_m) - \hat{y}^k(\tau_m)\|_{\infty} \quad [5.57]$$

for some $\gamma < 1$ and

$$\|H(b(\hat{x}_{past}, k))\hat{y}^k(\tau_m) - H(b(\hat{y}_{past}, k))\hat{y}^k(\tau_m)\|_{\infty} \leq \quad [5.58]$$

$$l_H \|b(\hat{x}_{past}, k) - b(\hat{y}_{past}, k)\|_{\infty} \|\hat{y}^k(\tau_m)\|_{\infty}$$

where l_H is the Lipschitz constant of H with respect to b .

Substituting Eqn. (5.57) and Eqn. (5.58) into Eqn. (5.55)

$$\|\hat{x}^{k+1}(\tau_m) - \hat{y}^{k+1}(\tau_m)\|_{\infty} \leq \gamma \|\hat{x}^k(\tau_m) - \hat{y}^k(\tau_m)\|_{\infty} + l_H M \|b(\hat{x}_{past}, k) - b(\hat{y}_{past}, k)\|_{\infty} \quad [5.59]$$

where $M = \max_m \|\hat{y}^k(\tau_m)\|_{\infty}$. Multiplying by e^{-Bmh} and taking the maximum over m

$$\max_m e^{-Bmh} \|\hat{x}^{k+1}(\tau_m) - \hat{y}^{k+1}(\tau_m)\|_{\infty} \leq \quad [5.60]$$

$$\gamma \max_m e^{-Bmh} \|\hat{x}^k(\tau_m) - \hat{y}^k(\tau_m)\|_{\infty} + l_H M \max_m e^{-Bmh} \|b(\hat{x}_{past}, k) - b(\hat{y}_{past}, k)\|_{\infty}.$$

Or, using Definition 5.1,

$$\|\{\hat{x}^{k+1}(\tau_m) - \hat{y}^{k+1}(\tau_m)\}\|_B \leq \|\{\hat{x}^k(\tau_m) - \hat{y}^k(\tau_m)\}\|_B + l_H M \|\{b(\hat{x}_{past}, k) - b(\hat{y}_{past}, k)\}\|_B \quad [5.61]$$

The term in Eqn. (5.60) $b(\hat{x}_{pass}^k, k) - b(\hat{y}_{pass}^k, k)$ can be expanded using the definition of b in Eqn. (5.48) to

$$h \sum_{j=1}^m \gamma_j [f_i(\hat{x}^{k+1,j}(\tau_{m-j}), u(\tau_{m-j})) - f_i(\hat{y}^{k+1,j}(\tau_{m-j}), u(\tau_{m-j}))]. \quad [5.62]$$

The β norm of the sequence whose terms are given in Eqn. (5.62) can be bounded using Lemma 5.2. That is,

$$\| \{ h \sum_{j=1}^m \gamma_j [f_i(\hat{x}^{k+1,j}(\tau_{m-j}), u(\tau_{m-j})) - f_i(\hat{y}^{k+1,j}(\tau_{m-j}), u(\tau_{m-j}))] \} \|_B \quad [5.63]$$

$$\leq hM \frac{e^{-Bhm}}{1 - e^{-Bhm}} \| f_i(\hat{x}^{k+1,j}(\tau_{m-j}), u(\tau_{m-j})) - f_i(\hat{y}^{k+1,j}(\tau_{m-j}), u(\tau_{m-j})) \|,$$

where M is the $\max_m \gamma_m$. Using the triangle inequality and the Lipschitz property of f ,

$$\| \{ h \sum_{j=1}^m \gamma_j [f_i(\hat{x}^{k+1,j}(\tau_{m-j}), u(\tau_{m-j})) - f_i(\hat{y}^{k+1,j}(\tau_{m-j}), u(\tau_{m-j}))] \} \|_B \leq \quad [5.64]$$

$$hMl \frac{e^{-Bhm}}{1 - e^{-Bhm}} \| \{ \hat{x}^{k+1}(\tau_m) - \hat{y}^{k+1}(\tau_m) \} \|_B + \| \{ \hat{x}^k(\tau_m) - \hat{y}^k(\tau_m) \} \|_B$$

where l is the Lipschitz constant of f with respect to x . This bound can be used in Eqn. (5.60) to yield

$$(1 - hMl \frac{e^{-Bhm}}{1 - e^{-Bhm}}) \| \{ \hat{x}^{k+1}(\tau_m) - \hat{y}^{k+1}(\tau_m) \} \|_B \leq \quad [5.65]$$

$$(\gamma - hMl \frac{e^{-Bhm}}{1 - e^{-Bhm}}) \| \{ \hat{x}^k(\tau_m) - \hat{y}^k(\tau_m) \} \|_B$$

or

$$\|\hat{x}^{k+1}(\tau_m) - \hat{y}^{k+1}(\tau_m)\|_B \leq \frac{(\gamma + hMl \frac{e^{-Bhm}}{1 - e^{-Bhm}})}{(1 - hMl \frac{e^{-Bhm}}{1 - e^{-Bhm}})} \|\hat{x}^k(\tau_m) - \hat{y}^k(\tau_m)\|_B \quad [5.66]$$

as $\gamma < 1$ there exists an h_0 and a $b > 0$ such that

$$\frac{(\gamma + hMl \frac{e^{-bhm}}{1 - e^{-bhm}})}{(1 - hMl \frac{e^{-bhm}}{1 - e^{-bhm}})} \leq 1 \quad [5.67]$$

for all $h < h_0$, which proves the theorem ■.

SECTION 5.3 - THE MULTI-RATE WR CONVERGENCE THEOREM

Theorem 5.1 suggests that the global-timestep discretized WR algorithm is not going to be any more efficient than the well-known relaxation-Newton algorithms described in Section 3.2, as the timestep constraints for the two methods are identical for the linear case. In fact, as Eqn. (5.10) indicates, WR is likely to be less efficient, because decomposition errors made in the first few timesteps propagate through the computations. That the discretized WR algorithm has proved to be more efficient in practice for some types of problems is because the discretized WR algorithm is intrinsically a multi-rate integration method. It is because this is the key aspect of the WR algorithm that the rest of this Chapter will be devoted to a proof that the discretized WR algorithm converges even when the timesteps for each subsystem are chosen independently.

Usually, choosing how to interpolate the discrete sequence of points produced by a numerical integration method is based only on what will produce attractive graphs of the computed solution. When multi-rate integration methods are applied to solving a system, interpolation plays a much more significant role. If two state variables in a system interact, and they are computed using different

timesteps, then to provide the value of one variable at the times required to compute the second variable, the first variable must be interpolated. In the case of WR, if the interpolation is not done carefully, convergence can be destroyed.

In this section, a convergence theorem for systems in normal form will be presented that demonstrates the key role of interpolation in the convergence of the multi-rate discretized WR algorithm. The theorem guarantees that the discretized WR algorithm is a contraction mapping assuming that the points produced by the numerical integration method are interpolated linearly. As the theorem proof will demonstrate, the linear interpolation has one particular property that aids convergence.

Consider the following system,

$$\dot{x}(t) = f(x(t), u(t)) \quad [5.68]$$

where $x(t) = (x_1(t), \dots, x_n(t))^T$, $x_i(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$, piecewise continuous, and $f = (f_1(x), \dots, f_n(x))^T$, $f_i: \mathbb{R}^n \rightarrow \mathbb{R}$ is Lipschitz continuous. If the Gauss-Seidel WR algorithm is applied to Eqn. (5.68), the iteration equation for x_i is

$$\dot{x}_i^{k+1}(t) = f_i(x_1^{k+1}(t), \dots, x_i^{k+1}(t), x_{i+1}^k(t), \dots, x_n^k(t), u(t)) \quad [5.69]$$

If Eqn. (5.69) is solved numerically using a multistep integration algorithm with a fixed-timestep h , the iteration equation becomes

$$\rho(\hat{x}_i^{k+1}(\tau_m)) = h\sigma(f_i(\hat{x}_i^{k+1}(\tau_m), \dots, \hat{x}_i^{k+1}(\tau_m), \hat{x}_{i+1}^k(\tau_m), \dots, \hat{x}_n^k(\tau_m), u(\tau_m))) \quad [5.70]$$

If different timesteps are used to solve the differential equations associated with the x_j^k variables, $i \neq j$, then Eqn. (5.70) makes no sense, because τ_m for the j^{th} equation may be different than τ_m for the i^{th} equation. In order to even write down the equations for the multi-rate case, some kind of interpolation operator must first be defined.

Definition 5.2: Given a finite sequence $\{y(\tau_m)\}$ of M elements, where $y(\tau_m) \in \mathbb{R}$, for all $m \leq M$, an interpolation function $I_t\{\bullet\}$ on the sequence is any function that maps the sequence and the independent variable $t \in \mathbb{R}$, $t \in [\tau_0, \tau_M]$ into \mathbb{R} , such that $I_t\{\bullet\}$ is continuous with respect to t , and that $I_{\tau_j}\{y(\tau_j)\} = y(\tau_j)$. ■

As an example, the linear interpolation of a sequence at a given time $t \in [\tau_0, \tau_M]$ would be

$$I_t\{y(\tau_m)\} = y(\tau_j) + \frac{y(\tau_{j+1}) - y(\tau_j)}{(\tau_{j+1} - \tau_j)}(t - \tau_j) \quad [5.71]$$

where j is such that $\tau_j \leq t \leq \tau_{j+1}$.

In order to write a form of Eqn. (5.70) for the multi-rate case, we will denote τ_m for the i^{th} equation as τ_m^i . Using this notation and the interpolation operator defined above, the unfortunately indice-filled iteration equation for \hat{x}_i^k for the multi-rate fixed-timestep case is

$$\rho(\hat{x}_i^{k+1}(\tau_m^i)) = h_i \sigma(f(I_{\tau_m^i}\{\hat{x}_1^{k+1}(\tau_m^1)\}, \dots, I_{\tau_m^i}\{\hat{x}_i^{k+1}(\tau_m^{i-1})\},$$

$$I_{\tau_m^i}\{\hat{x}_{i+1}^k(\tau_m^{i+1})\}, \dots, I_{\tau_m^i}\{\hat{x}_n^k(\tau_m^n)\}, u(\tau_m^i))$$

where h_i is the fixed-timestep for the i^{th} system. Using the inverse operator as in Eqn. (5.25),

$$\hat{x}_i^{k+1}(\tau_m^i) = h_i \rho^{-1} \sigma(f(I_{\tau_m^i}\{\hat{x}_1^{k+1}(\tau_m^1)\}, \dots, I_{\tau_m^i}\{\hat{x}_i^{k+1}(\tau_m^{i-1})\}, \quad [5.72]$$

$$I_{\tau_m^i}\{\hat{x}_{i+1}^k(\tau_m^{i+1})\}, \dots, I_{\tau_m^i}\{\hat{x}_n^k(\tau_m^n)\}, u(\tau_m^i))$$

The proof of Theorem 5.2 demonstrated that the fixed global-timestep discretized WR algorithm is a contraction mapping in an L_∞ norm on the sequence (see Definition 5.1). In the multi-rate case, this is not sufficient. Since interpolated as well as sequence values are used by subsystems, a

convergence proof must take into account the effect of the the interpolation on the sequence. The approach that will be used in the proof that follows is to view the multi-rate discretized WR algorithm, which necessarily includes an interpolation operator, as a map of continuous functions on $[0, T]$ to continuous functions. The implicitly defined map can be derived by applying the interpolation operator to both sides of Eqn 5.72 to yield

$$I_t\{\hat{x}_i^{k+1}(\tau_m^i)\} = I_t\{h\rho^{-1}\sigma(f(I_{\tau_m^i}\{\hat{x}_1^{k+1}(\tau_m^1)\}, \dots, I_{\tau_m^i}\{\hat{x}_i^{k+1}(\tau_m^{i-1})\}, \quad [5.73]$$

$$I_{\tau_m^i}\{\hat{x}_{i+1}^k(\tau_m^{i+1})\}, \dots, I_{\tau_m^i}\{\hat{x}_n^k(\tau_m^n)\}, u(\tau_m^i)\}.$$

To prove the convergence of the relaxation, the usual continuous-time L_β norm can be used,

$$\|x\|_\beta = \max_{[0, T]} e^{-\beta t} [\max_i |I_t\{x_i(\tau_m^i)\}|] \quad [5.74a]$$

or equivalently,

$$\|x\|_\beta = \max_i [\max_{[0, T]} e^{-\beta t} |I_t\{x_i(\tau_m^i)\}|], \quad [5.74b]$$

where x is used to denote the vector function on $[0, T]$ defined by $x(t) = (I_t\{x_1(\tau_m^1)\}, \dots, I_t\{x_n(\tau_m^n)\})^T$

Under certain conditions Eqn. (5.73) is a contraction map in the β norm of Eqn. (5.74). To prove this, Eqn. (5.73) will be applied to two sequences $\{x^k(\tau_m)\}$ and $\{y^k(\tau_m)\}$. The difference between Eqn. (5.73) applied to the two sequences is

$$I_t\{\hat{x}_i^{k+1}(\tau_m^i)\} - I_t\{\hat{y}_i^{k+1}(\tau_m^i)\} = \quad [5.75]$$

$$I_t\{h\rho^{-1}\sigma(f(I_{\tau_m^i}\{\hat{x}_1^{k+1}(\tau_m^1)\}, \dots, I_{\tau_m^i}\{\hat{x}_i^{k+1}(\tau_m^i)\}, I_{\tau_m^i}\{\hat{x}_{i+1}^k(\tau_m^{i+1})\}, \dots, I_{\tau_m^i}\{\hat{x}_n^k(\tau_m^n)\}, u(\tau_m^i))\} -$$

$$I_t\{h_i\rho^{-1}\sigma(f(I_{\tau_m}^i\{\hat{y}_1^{k+1}(\tau_m^1)\},\dots,I_{\tau_m}^i\{\hat{y}_i^{k+1}(\tau_m^i)\},I_{\tau_m}^i\{\hat{y}_{i+1}^k(\tau_m^{i+1})\},\dots,I_{\tau_m}^i\{\hat{y}_n^k(\tau_m^n)\},u(\tau_m^i))\}.$$

It is possible to simplify Eqn. (5.75) by limiting the type of interpolation operators to those that are linear functions of the sequence. To avoid confusion, by this it is not meant to limit consideration to only linear interpolation, but to those interpolation functions for which

$$I_t\{x(\tau_m)\} - I_t\{y(\tau_m)\} = I_t\{x(\tau_m) - y(\tau_m)\}$$

and

$$I_t\{\alpha x(\tau_m)\} = \alpha I_t\{x(\tau_m)\}$$

where $\{x(\tau_m)\}$, $\{y(\tau_m)\}$ are sequences in \mathbb{R} , and $\alpha \in \mathbb{R}$. For example, any of the spline or polynomial interpolation operators are linear functions of the sequence. Exploiting this linearity in Eqn. (5.75) leads to

$$I_t\{\hat{x}_i^{k+1}(\tau_m^i) - \hat{y}_i^{k+1}(\tau_m^i)\} = \quad [5.76]$$

$$h_i I_t\{ \rho^{-1} \sigma[(f(I_{\tau_m}^i\{\hat{x}_1^{k+1}(\tau_m^1)\},\dots,I_{\tau_m}^i\{\hat{x}_i^{k+1}(\tau_m^i)\},I_{\tau_m}^i\{\hat{x}_{i+1}^k(\tau_m^{i+1})\},\dots,I_{\tau_m}^i\{\hat{x}_n^k(\tau_m^n)\},u(\tau_m^i)) - (f(I_{\tau_m}^i\{\hat{y}_1^{k+1}(\tau_m^1)\},\dots,I_{\tau_m}^i\{\hat{y}_i^{k+1}(\tau_m^i)\},I_{\tau_m}^i\{\hat{y}_{i+1}^k(\tau_m^{i+1})\},\dots,I_{\tau_m}^i\{\hat{y}_n^k(\tau_m^n)\},u(\tau_m^i))] \}.$$

It is possible to show that the multi-rate discretized WR algorithm is a contraction mapping in the β norm of Eqn. (5.74) if the interpolation operator is limited to linear interpolation (as in Eqn. (5.71),) and the timesteps are made small enough.

Theorem 5.3: If the interpolation in Eqn. (5.73) is linear interpolation, then there exists a collection of timesteps $h_{i0} > 0$, $i \in \{1, \dots, n\}$ such that if $0 < h_i \leq h_{i0}$ for all i , then the multi-rate fixed-timestep discretized WR algorithm converges with the respect to the interpolated sequences.

The following simple Lemmas will be useful in the proof.

Lemma 5.3: If $I_t\{\bullet\}$ is the linear interpolation operator (as in Eqn. (5.73)), then given two sequences $\{x(\tau_m)\}$ and $\{y(\tau_m)\}$, if $x(\tau_i) \geq y(\tau_i)$ for all i then $I_t\{x(\tau_m)\} \geq I_t\{y(\tau_m)\}$ for all t for which the interpolation is defined. In addition, if $x_{\tau_m} = K$, $K \in \mathbb{R}$, for all $m \leq m'$, then $I_t\{x(\tau_m)\} = K$ for $t \leq \tau_{m'}$.

Lemma 5.3 follows directly from the definition of linear interpolation. As will be shown in the proof, this is the key property of linear interpolation with respect Theorem 5.3.

Lemma 5.4: If $I_t\{\bullet\}$ is the linear interpolation operator and $\{x(\tau_m)\}$ is a sequence in \mathbb{R} , then

$$\max_{[0,T]} e^{-Bt} |I_t\{\sum_{l=0}^{l=m_i} |\gamma_l| |I_{(m-l)h}\{x(\tau_m)\}|\}| \leq \frac{1}{1 - e^{-Bh}} M \max_{[0,T]} e^{-Bt} |I_t\{x(\tau_m)\}|$$

where $M = \max_l |\gamma_l|$

The proof of Lemma 5.4 parallels the arguments given in the proof of Lemma 5.2, and is omitted.

Proof of Theorem 5.3:

Expanding the $\rho^{-1}\sigma$ operator into its sum form leads to

$$I_t\{\hat{x}_i^{k+1}(\tau_m^i) - \hat{y}_i^{k+1}(\tau_m^i)\} =$$

$$h_i I_i \left\{ \sum_{l=0}^{l=m_i} \gamma_l [(f(I_{\tau(m-l)} \{\hat{x}_1^{k+1}(\tau_m^1)\}, \dots, I_{\tau(m-l)} \{\hat{x}_i^{k+1}(\tau_m^i)\}, \right.$$

$$I_{\tau(m-l)} \{\hat{x}_{i+1}^k(\tau_m^{i+1})\}, \dots, I_{\tau(m-l)} \{\hat{x}_n^k(\tau_m^n)\}, u(\tau_m^i))$$

$$- (f(I_{\tau(m-l)} \{\hat{y}_1^{k+1}(\tau_m^1)\}, \dots, I_{\tau(m-l)} \{\hat{y}_i^{k+1}(\tau_m^i)\},$$

$$I_{\tau(m-l)} \{\hat{y}_{i+1}^k(\tau_m^{i+1})\}, \dots, I_{\tau(m-l)} \{\hat{y}_n^k(\tau_m^n)\}, u(\tau_m^i))] \}.$$

Using Lemma 5.3 and the Lipschitz continuity property of f ,

$$|I_i \{\hat{x}_i^{k+1}(\tau_m^i) - \hat{y}_i^{k+1}(\tau_m^i)\}| \leq \quad [5.77]$$

$$|I_i \{ \sum_{l=0}^{l=m_i} |\gamma_l| [\sum_{j=1}^{j=i} h_i L_{ij} |I_{\tau(m-l)} \{\hat{x}_j^{k+1} - \hat{y}_j^{k+1}\}| + \sum_{j=i+1}^{j=n} h_i L_{ij} |I_{\tau(m-l)} \{\hat{x}_j^k - \hat{y}_j^k\}|] \} |$$

where L_{ij} is the Lipschitz constant of f_i with respect to x_j . Reorganizing, and exploiting the general linearity property of the interpolation operator and the triangle inequality,

$$|I_i \{\hat{x}_i^{k+1}(\tau_m^i) - \hat{y}_i^{k+1}(\tau_m^i)\}| \leq \quad [5.78]$$

$$\sum_{j=1}^{j=i} h_i L_{ij} |I_i \{ \sum_{l=0}^{l=m_i} |\gamma_l| |I_{\tau(m-l)} \{\hat{x}_j^{k+1} - \hat{y}_j^{k+1}\}| \} | +$$

$$\sum_{j=i+1}^{j=n} h_i L_{ij} |I_i \{ \sum_{l=0}^{l=m_i} |\gamma_l| |I_{\tau(m-l)} \{\hat{x}_j^k - \hat{y}_j^k\}| \} |$$

Multiplying by e^{-Bt} and taking maximums,

$$\max_{[0,T]} e^{-Bt} |I_t\{\hat{x}_i^{k+1}(\tau_m^i) - \hat{y}_i^{k+1}(\tau_m^i)\}| \leq \quad [5.79]$$

$$\sum_{j=1}^{j=i} h_i L_{ij} \max_{[0,T]} e^{-Bt} |I_t\{\sum_{l=0}^{l=n_i} |\gamma_l| |I_{\tau(m-l)}\{\hat{x}_j^{k+1} - \hat{y}_j^{k+1}\}| \}| +$$

$$\sum_{j=i+1}^{j=n} h_i L_{ij} \max_{[0,T]} e^{-Bt} |I_t\{\sum_{l=0}^{l=n_i} |\gamma_l| |I_{\tau(m-l)}\{\hat{x}_j^k - \hat{y}_j^k\}| \}|$$

Applying Lemma 5.4,

$$\max_{[0,T]} e^{-Bt} |I_t\{\hat{x}_i^{k+1}(\tau_m^i) - \hat{y}_i^{k+1}(\tau_m^i)\}| \leq \quad [5.80]$$

$$\left[\frac{Mh_i}{1 - e^{-Bh_i}} \sum_{j=1}^{j=i} L_{ij} \right] \|\hat{x}^{k+1} - \hat{y}^{k+1}\|_B + \left[\frac{Mh_i}{1 - e^{-Bh_i}} \sum_{j=i+1}^{j=n} L_{ij} \right] \|\hat{x}^k - \hat{y}^k\|_B$$

where $\|\hat{x}^k - \hat{y}^k\|_B$ is the L_B norm defined in Eqn. (5.74b).

For any $\delta > 0$ there exists a collection of steps $\{h_{i_0}, \dots, h_{n_0}\}$, all strictly positive, and a $B > 0$ such that

$$\delta > \frac{Mh_i \sum_{j=1}^{j=i} L_{ij}}{1 - e^{-Bh_i}} \quad [5.81]$$

for all $h_i \leq h_{n_0}$, for all i . Substituting into Eqn 5.80,

$$\max_{[0,T]} e^{-Bt} |I_t\{\hat{x}_i^{k+1}(\tau_m^i) - \hat{y}_i^{k+1}(\tau_m^i)\}| \leq \delta \|\hat{x}^{k+1} - \hat{y}^{k+1}\|_B + \delta \|\hat{x}^k - \hat{y}^k\|_B \quad [5.82]$$

Since Eqn. (5.81) holds for all i ,

$$\|\hat{x}^{k+1} - \hat{y}^{k+1}\|_B \leq \delta \|\hat{x}^{k+1} - \hat{y}^{k+1}\|_B + \delta \|\hat{x}^k - \hat{y}^k\|_B \quad [5.83]$$

Reorganizing,

$$\|\hat{x}^{k+1} - \hat{y}^{k+1}\|_B \leq \frac{\delta}{1-\delta} \|\hat{x}^k - \hat{y}^k\|_B \quad [5.84]$$

Clearly, there exists a $\delta > 0$ such that Eqn. (5.84) is a contraction ($\frac{\delta}{1-\delta} < 1$). Let $\hat{\delta}$ be that δ . Since there exists a $B > 0$ and collection of h_0 's > 0 such that Eqn. (5.84) holds for $\delta = \hat{\delta}$ for all $0 < h_i \leq h_0$, the theorem is proved. ■

Perhaps the most surprising aspect of the proof of Theorem 5.3 is that the ratio of the timesteps from one system to the next does *not* seem to play a role. This is an extremely important observation given that the discretized WR algorithm was developed to allow different subsystems to take vastly different timesteps. If a large ratio between timesteps destroyed the WR convergence, then the applicability of the WR algorithm to multi-rate problems would be limited.

A second important observation is that the only property of linear interpolation used in the course of the proof was that stated in Lemma 5.3. Therefore, other interpolations that have this property will work as well. Higher order polynomial interpolation functions do not have the property stated in Lemma 5.3, but as they are substantially more accurate than linear interpolation, they are extremely useful. An extension of the above theorem to general polynomial interpolation does not seem to be straight-forward, and may call for an entirely different approach.

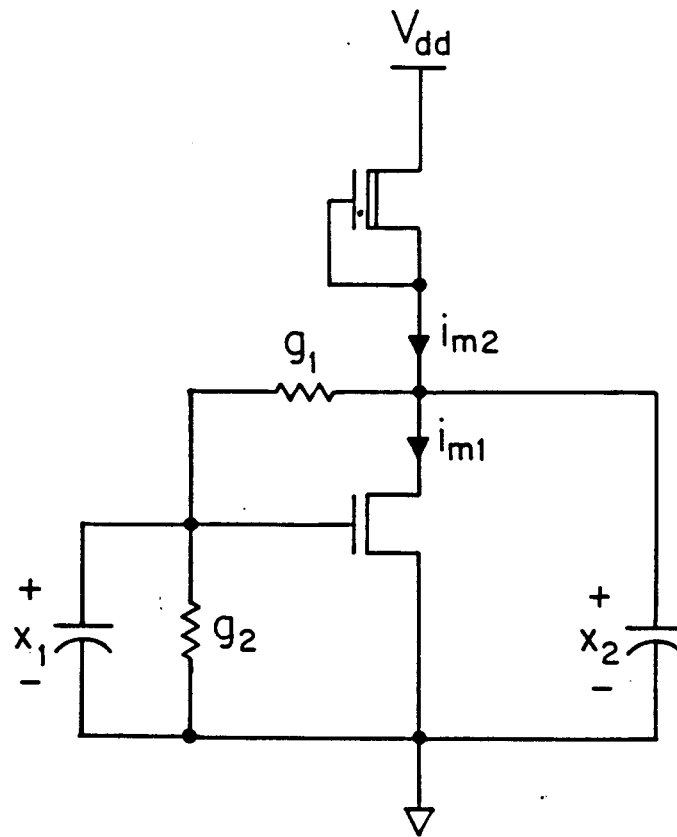


Figure 5.1 - Two Node Inverter Circuit with Feedback

CHAPTER 6 - ACCELERATING WR CONVERGENCE

In Chapter 7, several techniques used to improve the efficiency and robustness of the WR algorithm when applied to simulating MOS circuits will be described. In this chapter the theoretical background for two of these techniques will be presented. We will first analyze nonuniformity in WR convergence, which explains why breaking the simulation interval into pieces, called windows, can be used to reduce the number of relaxation iterations required to achieve convergence. Then we will consider how to partition large systems into subsystems in such a way that the WR algorithm will converge rapidly.

SECTION 6.1 - UNIFORMITY OF WR CONVERGENCE

The convergence theorem presented in Section 3.2 guarantees that the WR algorithm is a contraction mapping in an exponentially weighted norm. In this section, we will examine the implications of this choice of norm. First, the WR algorithm will be applied to two example problems to exhibit the different manners in which the algorithm converges. We will then prove that for a special class of systems WR converges in a uniform manner, or formally, that WR is a contraction in an unweighted norm for any time interval. Because most circuit problems do not generate systems in this special class, we will prove that the WR algorithm is a contraction in an unweighted norm for any system for which the WR algorithm converges, if the time interval is made short enough. This suggests that the number of iterations required to achieve WR convergence can be reduced by breaking the simulation interval into short pieces, and in Chapter 7 we will present an adaptive algorithm that attempts to exploit this property of WR.

Consider the following nonlinear ordinary differential equation in $x_1(t), x_2(t) \in \mathbb{R}$ with input $u \in \mathbb{R}$ that approximately describes the cross-coupled *nor* logic gate in Fig. 6.1a (the approximate equations represent a normalization that converts the simulation interval $[0, T]$ to $[0, 1]$).

$$\dot{x}_1 = (5 - x_1) - x_1(x_2)^2 - 5x_1u \quad [6.1]$$

$$\dot{x}_2 = (5 - x_2) - x_2(x_1)^2$$

$$x_1(0) = 5.0 \quad x_2(0) = 0.0$$

The Gauss-Seidel WR Algorithm given in Section 1.2 was used to solve the for the behavior of the cross-coupled *nor* gate circuit approximated by the above small system of equations. In Fig. 6.1b plots of the input $u(t)$, the exact solution for $x_1(t)$, and the relaxation iteration waveforms for $x_1(t)$ for the 5th, 10th and 20th iterations are shown. The plots demonstrate a property typical of the WR algorithm when applied to systems with strong coupling: the difference between the iteration waveforms and correct solution is not reduced at every time point in the waveform. Instead, each iteration lengthens the interval of time, starting from zero, for which the waveform is close to the exact solution.

As an example of "better" convergence, consider the following differential equation in x_1, x_2, x_3 with input u that approximately describes the shift register in Fig. 6.2a (here the simulation interval $[0,7]$ has been normalized to $[0,1]$)

$$\dot{x}_1 = (5.0 - x_1) - x_1(u)^2 - (x_1 - x_2) \quad [6.2]$$

$$\dot{x}_2 = (x_1 - x_2)$$

$$\dot{x}_3 = (5.0 - x_3) - x_3(x_2)^2$$

$$x(0) = 0.$$

The Gauss-Seidel WR Algorithm given in Section 1.2 was used to solve the original system approximated by the above system of equations. The input $u(t)$, the exact solution for $x_1(t)$, and the

waveforms for $x_1(t)$ computed from the first, second, and third iterations of the WR algorithm are plotted in Fig. 6.2b. As the plots for this example show, the difference between the iteration waveforms and the correct solution is reduced throughout the entire waveform.

Perhaps surprisingly, the behavior of the first example is consistent with the WR convergence theorem, even though that theorem states that the iterations converge uniformly. This is because it was proved that the WR method is a contraction map in the following nonuniform norm on $C([0, T], \mathbb{R}^n)$:

$$\max_{[0, T]} e^{-bt} \|f(t)\|$$

where $b > 0$, $f(t) \in \mathbb{R}^n$, and $\|\cdot\|$ is a norm on \mathbb{R}^n . Note that $\|f(t)\|$ can increase as e^{bt} without increasing the value of this function space norm. If $f(t)$ grows slowly, or is bounded, it is possible to reduce the function space norm by reducing $\|f(t)\|$ only on some small interval in $[0, T]$, though it will be necessary to increase this interval to further decrease the function space norm. The waveforms in the more slowly converging example above, converge in just this way; the function space norm is decreased after every iteration of the WR algorithm because significant errors are reduced over larger and larger intervals of time.

The examples above lead to the following definition:

Definition 6.1: A differential system of the form given in Eqn. (2.2) said to have *strict WR contractivity property* on $[0, T]$, if the WR algorithm applied to the system contracts in a uniform norm on $[0, T]$, i.e.

$$\max_{[0, T]} \|x^{k+1}(t) - x^k(t)\| < \max_{[0, T]} \|x^k(t) - x^{k-1}(t)\| \quad [6.3]$$

where $x^k(t) \in \mathbb{R}^n$ on $t \in [0, T]$ is the k 'th iterate of Algorithm 4.1 and $\|\cdot\|$ is any norm on \mathbb{R}^n . If the WR algorithm applied to the system is a contraction in a uniform norm on $[0, T]$ for any $T > 0$ then we say that the system has the strict WR contractivity property on $[0, \infty)$ ■

For a system of equations to have the strict WR contractivity property on $[0, \infty)$ it must be more than just loosely coupled. In addition, the decomposed equations solved at each iteration of the waveform relaxation must be well-damped, so that errors due to the decomposition die off in time, instead of accumulating or growing. As an example, we will prove that a system in normal form,

$$\dot{x}(t) = f(x(t), u(t)) \quad x(0) = x_0 \quad [6.4]$$

where $x(t) \in \mathbb{R}^n$ on $t \in [0, T]$; $u(t) \in \mathbb{R}^r$ on $t \in [0, T]$ piecewise continuous; and $f: \mathbb{R}^n \times \mathbb{R}^r \rightarrow \mathbb{R}^n$ is globally Lipschitz continuous will have the strict WR contractivity property on $[0, T]$ for any $T < \infty$ if f has a property we refer to as diagonally dominant negative monotonicity. This property, which we define precisely below, just implies that the original system is loosely coupled, and the decomposed equations generated by a WR algorithm are well-damped (A similar result in a different setting can be found in [61]).

Definition 6.2: Let $f(x, u)$ be a continuous map from $\mathbb{R}^n \times \mathbb{R}^r \rightarrow \mathbb{R}^n$ where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^r$ and f is globally Lipschitz continuous with respect to x for all $u \in \mathbb{R}^r$. f is said to be negative monotone if there exists a positive number λ such that

$$(x - y) \cdot (f(x, u) - f(y, u)) \leq -\lambda(x - y) \cdot (x - y) \quad [6.5]$$

(here \cdot indicates a scalar product) for all $x, y \in \mathbb{R}^n$ and $u \in \mathbb{R}^r$. Let $v^i \in \mathbb{R}^n$ be the i^{th} unit vector. Then f is said to be diagonally negative monotone if there exists a collection of positive numbers λ_i such that

$$v^i \cdot (f(x + \varepsilon v^i, u) - f(x, u)) \leq -\lambda_i \varepsilon^2 \quad [6.6]$$

for any positive $\varepsilon \in \mathbb{R}$, $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^r$. If f is globally Lipschitz continuous, there exist positive numbers l_{ij} , $i, j \in [1, \dots, n]$ such that for any $\varepsilon \in \mathbb{R}$

$$\|v^i \cdot (f(x + \varepsilon v^i, u) - f(x, u))\| \leq l_{ij} |\varepsilon| \quad [6.7]$$

for all $x \in \mathbb{R}^n$, $u \in \mathbb{R}^r$. A mapping f , is a diagonally dominant negative monotone if f is a strictly diagonally negative monotone and $\lambda_i > \sum_{j \neq i} l_{ij}$, where λ_i is as in Eqn. (6.6). (This is a stricter definition than in previous literature[30]).

To prove the theorem about diagonally dominant negative monotone systems we will use the following lemma.

Lemma 6.1: Let k , $x(t)$, $\dot{x}(t) \in \mathbb{R}$ be such that

$$x(t)\dot{x}(t) \leq -\lambda x(t)x(t) + kx(t) \quad x(0) = 0 \quad [6.8]$$

for some positive number λ . Then $|x(t)| < |k|\lambda^{-1}$ for all $t > 0$.

Proof of Lemma 6.1:

Substituting $\frac{d}{dt}|x(t)|^2$ for $2x(t)\dot{x}(t)$ in Eqn. (6.8) and taking absolute values

$$\frac{d}{dt}|x(t)|^2 \leq -2\lambda|x(t)|^2 + 2|k||x(t)|$$

Therefore, $\frac{d}{dt}|x(t)| \leq -\lambda|x(t)| + |k|$ or $|x(t)| = 0$. This implies, by a theorem in differential inequalities[39], that

$$|x(t)| \leq \frac{|k|}{\lambda}(1 - e^{-\lambda t}) \leq \frac{|k|}{\lambda}$$

which proves the lemma. ■

We now prove the theorem.

Theorem 6.1: Let a system of equations of the form of Eqn. (6.4) be such that $f(x,u)$ is diagonally dominant negative monotone. Then the system has the strict WR contractivity property on $[0, T]$ for all $T < \infty$. ■

Proof of Theorem 6.1:

Again we will only present the proof for the Gauss-Seidel case, but the result holds for the Gauss-Jacobi case also. The iteration equations for the Gauss-Seidel WR algorithm applied to Eqn. (6.4) are, for each $i \in [1, \dots, n]$,

$$x_i^{k+1} = f_i(x_1^{k+1}, \dots, x_i^{k+1}, x_{i+1}^k, \dots, x_n^k, u) \quad [6.9]$$

where x , u , and f are functions of time, but the dependence on time has been dropped for notational convenience. Taking the difference between the k and $k + 1$ iteration for each $i \in [1, \dots, n]$ yields:

$$x_i^{k+1} - x_i^k = f_i(x_1^{k+1}, \dots, x_i^{k+1}, x_{i+1}^k, \dots, x_n^k, u) - f_i(x_1^k, \dots, x_i^k, x_{i+1}^{k-1}, \dots, x_n^{k-1}, u)$$

Multiplying through by $x_i^{k+1} - x_i^k$ and using the Lipschitz and diagonal negative monotone properties of f we get

$$(x_i^{k+1} - x_i^k)(x_i^{k+1} - x_i^k) \leq -\lambda_i(x_i^{k+1} - x_i^k)(x_i^{k+1} - x_i^k) + \quad [6.10]$$

$$\sum_{j=1}^{i-1} l_{ij} |x_j^{k+1} - x_j^k| \cdot |x_i^{k+1} - x_i^k| + \sum_{j=i+1}^n l_{ij} |x_j^k - x_j^{k-1}| \cdot |x_i^{k+1} - x_i^k|$$

where $|x_i|$ denotes the absolute value of x_i , and l_{ij} and λ_i are as in Definition 6.2. From the estimate in Lemma 6.1,

$$|x_i^{k+1} - x_i^k| < \sum_{j=1}^{i-1} l_{ij} \lambda_i^{-1} |x_j^{k+1} - x_j^k| + \sum_{j=i+1}^n l_{ij} \lambda_i^{-1} |x_j^k - x_j^{k-1}| \quad [6.11]$$

Let $A \in \mathbb{R}^{n \times n}$ be a matrix defined by $A_{ij} (i \neq j) = l_{ij} \lambda_i^{-1}$ and $A_{ii} = 0$. Then $A = L + U$ where L is strictly lower triangular, U is strictly upper triangular. Rewriting Eqn. (6.11) in matrix form

$$(I - L) |x^{k+1} - x^k| \leq U |x^k - x^{k-1}| \quad [6.12]$$

where $|x|$ is the vector whose elements are the absolute value of the elements of x , and the inequality holds for each element-by-element comparison. To show that Eqn. (6.12) implies $\|x^{k+1} - x^k\|_\infty < \|x^{k+1} - x^k\|_\infty$ requires slightly complicated argument, as the inequality will not still hold if both sides of Eqn. (6.12) are multiplied by $(I - L)^{-1}$. Since $(I - L)$ is diagonally dominant with unity on the diagonal and negative lower triangular off-diagonal entries, if r is a solution to $(I - L)|r| = U|x^k - x^{k-1}|$ then $|r| \geq |x^{k+1} - x^k|$. Given that f is diagonally dominant, $\|(I - L)^{-1}U\|_\infty < 1$ (Lemma 4.2), from which it follows that $|x^{k+1} - x^k| \leq r < |x^k - x^{k-1}|$. Then from Eqn. (6.12) we get

$$\max_{[0,T]} \|x^{k+1}(t) - x^k(t)\|_\infty < \max_{[0,T]} \|x^k(t) - x^{k-1}(t)\| \quad [6.13]$$

for any $T < \infty$, which proves the theorem. ■

As the crossed *rand* gate example indicates, many systems of interest do not have the strict WR contractivity property on $[0, T]$ for all $T < \infty$. However, we will prove that any system that satisfies the WR convergence theorem will also have the strict WR contractivity property on some nonzero interval.

Theorem 6.2: For any system of the form of Eqn. (2.2) which satisfies the assumptions of the WR convergence theorem (Theorem 4.1) there exists a $T > 0$ such that the system has the strict WR contractivity property on $[0, T]$.

Proof of Theorem 6.2

We prove the theorem only for the Gauss-Seidel WR algorithm but, as before, the theorem also holds for the Gauss-Jacobi case. Starting with Eqn. (4.8) and substituting x^k for x' ,

$$\dot{x}^{k+1}(t) - \dot{x}^k(t) = \quad [6.14]$$

$$(L_{k+1}(t) + D_{k+1}(t))^{-1} U_{k+1}(t) \dot{x}^k(t) - (L_k(t) + D_k(t))^{-1} U_k(t) \dot{x}^{k-1}(t) +$$

$$(L_{k+1}(t) + D_{k+1}(t))^{-1} \hat{f}(x^{k+1}, x^k, u) - (L_k(t) + D_k(t))^{-1} \hat{f}(x^k, x^{k-1}, u)$$

To simplify the notation, let $A_k(t)$, $B_k(t) \in \mathbb{R}^{n \times n}$ be defined by $A_k(t) = (L_k(t) + D_k(t))^{-1} U_k(t)$, $B_k(t) = (L_k(t) + D_k(t))^{-1}$. It is important to keep in mind that $(L_k(t) + D_k(t))^{-1} U_k(t)$, and $(L_k(t) + D_k(t))^{-1}$ are functions of x^k , and by definition, so are $A_k(t)$ and $B_k(t)$. Expanding the above equation and integrating,

$$\int_0^t (\ddot{x}^{k+1}(\tau) - \ddot{x}^k(\tau)) d\tau = \int_0^t A_{k+1}(\tau) (\dot{x}^k(\tau) - \dot{x}^{k-1}(\tau)) d\tau + \quad [6.15]$$

$$\int_0^t [A_{k+1}(\tau) - A_k(\tau)] \dot{x}^{k-1}(\tau) d\tau +$$

$$\int_0^t B_{k+1}(\tau) [\hat{f}(x^{k+1}(\tau), x^k(\tau), u(\tau)) - \hat{f}(x^k(\tau), x^{k-1}(\tau), u(\tau))] d\tau +$$

$$\int_0^t [B_{k+1}(\tau) - B_k(\tau)] \hat{f}(x^k(\tau), x^{k-1}(\tau), u(\tau)) d\tau$$

Integrating by parts and using the fact that $x^k(0) - x^{k-1}(0) = 0$,

$$x^{k+1}(t) - x^k(t) = A_{k+1}(t) [x^k(t) - x^{k-1}(t)] - \quad [6.16]$$

$$\int_0^t \frac{d}{d\tau} A_{k+1}(\tau) [x^k(\tau) - x^{k-1}(\tau)] d\tau + \int_0^t [A_{k+1}(\tau) - A_k(\tau)] \dot{x}^{k-1} d\tau +$$

$$\int_0^t B_{k+1}(\tau) [\hat{f}(x^{k+1}(\tau), x^k(\tau), u(\tau)) - \hat{f}(x^k(\tau), x^{k-1}(\tau), u(\tau))] d\tau +$$

$$\int_0^t [B_{k+1}(\tau) - B_k(\tau)] \hat{f}(x^k(\tau), x^{k-1}(\tau), u(\tau)) d\tau$$

Taking norms, and using the Lipschitz continuity of $f, A_k(t)$, and $B_k(t)$, and the uniform boundedness of $B_k(t)$ in x (see Theorem 4.1):

$$\|x^{k+1}(t) - x^k(t)\| = \int_0^t (l_1 K + k_1 \tilde{M} + k_3 \tilde{N}) \|x^{k+1}(\tau) - x^k(\tau)\| d\tau \leq \quad [6.17]$$

$$\gamma \|x^k(t) - x^{k-1}(t)\| + \int_0^t (l_2 K + k_1 \tilde{M} + 2k_2 \tilde{M} + k_4 \tilde{N}) \|x^k(\tau) - x^{k-1}(\tau)\| d\tau$$

where l_1, l_2 are the Lipschitz constants of \hat{f} with respect to x^{k+1} and x^k respectively; k_1, k_2, k_3, k_4 are the Lipschitz constants for $A_{k+1}(t), B_{k+1}(t)$ with respect to their x_{k+1} and x_k arguments respectively; $\gamma = \max_{x,k} [(L_k + D_k) U_k] < 1$; and \tilde{M} and \tilde{N} are the *a priori* bounds on \dot{x}^k and \hat{f} found in the proof of Theorem 4.1. Note that $\frac{d}{d\tau} A_{k+1}(\tau) = \frac{d}{dx^{k+1}} A_{k+1}(\tau) \dot{x}^{k+1} + \frac{d}{dx^k} A_{k+1}(\tau) \dot{x}^k < k_1 \tilde{M} + k_2 \tilde{M}$. Moving the \max (over t) norms outside the integrals and integrating yields

$$\max_{[0,T]} \|x^{k+1}(t) - x^k(t)\| \leq \quad [6.18]$$

$$\frac{\gamma + T(l_2 K + k_1 \tilde{M} + 2k_2 \tilde{M} + k_4 \tilde{N})}{1 - T(l_1 K + k_1 \tilde{M} + k_3 \tilde{N})} \max_{[0,T]} \|x^k(t) - x^{k-1}(t)\|.$$

Since $\gamma < 1$, a $T' > 0$ exists such that
$$\frac{\gamma + T' (l_2 K + k_1 \tilde{M} + 2k_2 \tilde{M} + k_4 \tilde{N})}{1 - T' (l_1 K + k_1 \tilde{M} + k_3 \tilde{N})} = \alpha < 1 .$$

With this T' , Eqn. (6.17) becomes

$$\max_{[0, T']} \|x^{k+1} - x^k\| \leq \alpha \max_{[0, T']} \|x^k - x^{k-1}\| \quad [6.19]$$

for $\alpha < 1$, which proves the theorem. ■

Theorem 6.2 guarantees that the WR algorithm is a contraction mapping in a uniform norm for any system, provided the interval of time over which the waveforms are computed is made small enough. This suggest that the interval of simulation $[0, T]$ should be broken up into *windows*, $[0, T_1]$, $[T_1, T_2]$, ..., $[T_{n-1}, T_n]$ where the size of each window is small enough so that the WR algorithm contracts uniformly throughout the entire window. The smaller the window is made, the faster the convergence. However, as the window size becomes smaller, the advantages of the waveform relaxation are lost. Scheduling overhead increases when the windows become smaller, since each subsystem must be processed at each iteration in every window. If the windows are made very small, timesteps chosen to calculate the waveforms are limited by the window size rather than by the local truncation error, and the advantages of the multi-rate nature of WR will be lost.

The lower bound for the region over which WR contracts uniformly given in Theorem 6.2 is too conservative in most cases to be of direct practical use. As mentioned above, in order for the WR algorithm to be efficient it is important to pick the largest windows over which the iterations actually contract uniformly, but the theorem only provides a worst-case estimate. Since it is difficult to determine *a priori* a reasonable window size to use for a given nonlinear problem, window sizes are usually determined dynamically, by monitoring the computed iterations(See Chapter 7)[18]. Since Theorem 6.2 guarantees the convergence of WR over *any* finite interval, a dynamic scheme does not have to pick the window sizes very accurately. The only cost of a bad choice of window is loss of efficiency, the relaxation *still* converges.

SECTION 6.2 - PARTITIONING LARGE SYSTEMS

In Algorithm 4.1, the system equations are solved as single differential equations in one unknown, and these solutions are iterated until convergence. If this kind of node-by-node decomposition strategy is used for systems with even just a few tightly coupled nodes, the WR algorithm will converge very slowly. As an example, consider the three node circuit in Fig. 6.3a, a two inverter chain separated by a resistor-capacitor network. In this case, the resistor-capacitor network is intended to model wiring delays, so the resistor has a large conductance compared to the other conductances in the circuit. The current equations for the system can be written down by inspection and are:

$$C\dot{x}_1 + i_{m1}(x_1, vdd) + i_{m2}(x_1, u) + g(x_1 - x_2) = 0 \quad [6.20]$$

$$C\dot{x}_2 g(x_2 - x_1) = 0$$

$$C\dot{x}_3 i_{m3}(x_3, x_2) + i_{m4}(x_3, vdd) = 0$$

Linearizing and normalizing time (so that the simulation interval $[0, T]$ is converted to $[0, 1]$) yields a 3x3 linear equation:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -10 & 9.5 & 0 \\ 9.5 & -9.5 & 0 \\ 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix} \quad [6.21]$$

$$x_1(0) = x_2(0) = 0 \quad x_3(0) = 5$$

Algorithm 4.1 was used to solve the original nonlinear system. The input $u(t)$, the exact solution for x_2 , and the first, fifth and tenth iteration waveforms generated by the WR algorithm for x_2 are plotted in Fig. 6.3b. As the plot indicates, the iteration waveforms for this example are converg-

ing very slowly. The reason for this slow convergence can be seen by examining the linearized system. It is clear x_1 and x_2 are tightly coupled by the small resistor modeling the wiring delay.

If Algorithm 4.1 is modified, so that x_1 and x_2 are lumped together and solved directly, we get the following iteration equations:

$$\begin{bmatrix} \dot{x}_1^{k+1} \\ \dot{x}_2^{k+1} \end{bmatrix} = \begin{bmatrix} -10 & 9.5 & 0 \\ 9.5 & -9.5 & 0 \end{bmatrix} \begin{bmatrix} x_1^{k+1} \\ x_2^{k+1} \\ x_3^k \end{bmatrix} + \begin{bmatrix} 5 \\ 0 \end{bmatrix} \quad [6.22]$$

$$\dot{x}_3^{k+1} = -x_2^{k+1} - x_3^{k+1}$$

The modified WR algorithm now converges in one iteration, because x_3 only depends on the "block" of x_1 and x_2 , and that block is independent of x_3 .

As the example above shows, lumping together tightly coupled nodes and solving them directly can greatly improve the efficiency of the WR algorithm. For this reason, the first step in almost every WR-based program is to *partition* the system, to scan all the nodes in the system and determine which should be lumped together and solved directly. Partitioning "well" is difficult for several reasons. If too many nodes are lumped together, the advantages of using relaxation will be lost, but if any tightly coupled nodes are not lumped together then the WR algorithm will converge very slowly. And since the aim of WR is to perform the simulation rapidly, it is important that the partitioning step not be computationally expensive.

Several approaches have been applied to solve this partitioning problem. The first approach is to require the user to partition the system[15]. This technique is reasonable for the simulation of large digital integrated circuits because usually the large circuit has already been broken up into small, fairly independent pieces to make the design easier to understand and manage. However, what is a sensible partitioning from a design point of view may not be a good partitioning for the WR algorithm. For this reason programs that require the user to partition the system sometimes perform a "sanity

check" on the partitioning. A warning is issued if there are tightly coupled nodes that have not been lumped together.

A second approach to partitioning, also tailored to digital integrated circuits, is the functional extraction method[16]. In this method the equations that describe the system are carefully examined to try to find functional blocks (i.e. a *nand* gate or a *flip-flop*). It is then assumed that nodes of the system that are members of the same functional block are tightly coupled, and are therefore grouped together. This type of partitioning is difficult to perform, since the algorithm must recognize broad classes of functional blocks, or nonstandard blocks may not be treated properly. However, the functional extraction method can produce very good partitions because the relative importance of the coupling of the nodes can be accurately estimated.

Since it is the intent of the partitioning to improve the speed of convergence of the relaxation, it is sensible to partition a large circuit with this, rather than topology or functionality, in mind. In this section we will develop an algorithm based on this idea. As it is difficult to get estimates of the speed of WR convergence directly, We will start with an exact analysis of a relaxation algorithm applied to a simple 2x2 linear algebraic example, and then lift the result to a heuristic for partitioning large linear algebraic problems. Then a relationship will be established between the convergence speed of the linear WR algorithm, and that of two linear algebraic problems.

The following definition will be useful for describing the rate of convergence of relaxation algorithms.

Definition 6.3: Let $x^k \in \mathbb{R}^n$ be generated by the k^{th} iteration of an algebraic relaxation algorithm applied to a system of the form $f(x) = 0$, where $x \in \mathbb{R}^n$ and $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$. Then the L_∞ iteration factor γ_∞ is defined as the smallest positive number such that

$$\|x^{k+1} - x^k\|_\infty \leq \gamma_\infty \|x^k - x^{k-1}\|_\infty$$

for any $k > 0$, and any bounded initial guess x^0 ■.

Since the difference between the exact solution, x , and the result of the k^{th} step of a relaxation, x^k , is less than $(\gamma_{\infty})^k \|x - x^0\|_{\infty}$, the size of γ_{∞} is an indication of how fast the relaxation converges. If γ_{∞} is much less than 1 then the relaxation is certain to converge rapidly, but if $\gamma_{\infty} \geq 1$ the relaxation may not converge, and if γ_{∞} is close to 1 the convergence may be very slow.

Consider the simple 2x2 matrix problem,

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad [6.23]$$

If the Gauss-Jacobi relaxation algorithm is used to solve Eqn. (6.23) (See Section 3.2) then the L_{∞} iteration factor is the L_{∞} induced norm of

$$\begin{bmatrix} 0 & \frac{a_{12}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & 0 \end{bmatrix} \quad [6.24a]$$

which is

$$\gamma_{\infty}^{GJ} = \max\left(\frac{a_{12}}{a_{11}}, \frac{a_{21}}{a_{22}}\right) \quad [6.24b]$$

and if the Gauss-Seidel relaxation algorithm is used, then the L_{∞} iteration factor is the L_{∞} induced norm of

$$\begin{bmatrix} 0 & \frac{1}{a_{11}} \\ 0 & \frac{a_{21}a_{12}}{a_{11}a_{22}} \end{bmatrix}, \quad [6.25a]$$

which is

$$\gamma_{\infty}^{GS} = \left| \frac{a_{21}a_{12}}{a_{11}a_{22}} \right| \quad [6.25b]$$

For the 2x2 linear system of Eqn. (6.23), Eqn. (6.24b) and Eqn. (6.25b) can be used to decide whether to use relaxation, or lump the two nodes together and use direct methods. The criteria that γ_{ij}^2 be small (much less than one), which we will refer to as the *diagonally dominant loop* criteria, has proved to be a useful heuristic for partitioning the large systems generated by circuit problems. For the linear algebraic problem

$$Ax = b \quad [6.26]$$

where $x = (x_1, \dots, x_n)^T$, $b = (b_1, \dots, b_n)^T$, $x_i, b_i \in \mathbb{R}$, $A \in \mathbb{R}^{n \times n}$, invertible, $A = (a_{ij})$, we have the following partitioning algorithm.

Algorithm 6.1 Diagonal Dominant Loop Partitioning for $Ax = b$

```

for all ( i,j in N ) {
    if (  $\frac{a_{ij}a_{ji}}{a_{jj}a_{ii}} > \alpha$  ) {  $x_i$  is lumped with  $x_j$  }
}

```

■

The constant α is dependent on the problem, and is roughly related to the desired L_∞ iteration factor, so the smaller α is made, the more likely nodes are to be lumped together.

Although Algorithm 6.1 works well for the matrices generated by a wide variety of circuit problems, it is only a heuristic. There are circuit examples for which the diagonally dominant loop criteria does not indicate tightly coupled nodes that should be placed in the same partition. A particularly common circuit example for which Algorithm 6.1 does not lump tightly coupled nodes together is given in Fig. 6.4, an inverter driving a series of resistors. This is just a more complex version of the example given at the beginning of this section. The KCL equations for the circuit, approximating the inverter's output as a one volt voltage source,

$$0.01x_1 + 10.0(x_1 - x_2) = 0.01$$

$$10.0(x_2 - x_1) + 1.0(x_2 - x_3) = 0$$

$$1.0(x_3 - x_2) + 10.0(x_3 - x_4) = 0$$

$$0.01x_4 + 10.0(x_4 - x_3) = 0$$

or in matrix form,

$$\begin{bmatrix} 10.01 & -10.0 & 0.0 & 0.0 \\ -10.0 & 11.0 & -1.0 & 0.0 \\ 0.0 & -1.0 & 11.0 & -10.0 \\ 0.0 & 0.0 & -10.0 & 10.01 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad [6.27]$$

If Algorithm 6.1 is used to partition the matrix in Eqn. (6.27) and $\alpha = 0.1$, then x_1 will be lumped with x_2 and x_3 will be lumped with x_4 . The spectral radius for the iteration matrix generated by applying block Gauss-Seidel relaxation to the partitioned subsystems is ≈ 0.98 . The spectral radius is a lower bound on the iteration factor in any norm. Since it is very close to one, the relaxation will converge slowly.

The reason the diagonally dominant loop criteria sometimes produces misleading results is that it is too local a criterion, it only indicates how mutually coupled two nodes are, compared to how coupled they are to other nodes in the problem. If two nodes are extremely tightly coupled as are the pairs x_1, x_2 and x_3, x_4 in the example of Eqn. (6.27), then each of the nodes in the pair will appear relatively loosely coupled to other nodes in the problem, even if they are tightly enough coupled to other nodes to slow the relaxation.

It is possible to modify the diagonally dominant loop partitioning algorithm so that it will produce good partitions for problems which contain subsystems like the example of Eqn. (6.27). To demonstrate the algorithm, we consider a different approach to partitioning. Consider a problem of

the form of Eqn. (6.26), $Ax - b = 0$, and define $\lambda_i = \frac{\partial x_i}{\partial b_i}$, which is just the i^{th} diagonal term of A^{-1} . Then new algorithm is generated by replacing $\frac{1}{a_{ii}}$ with λ_{ii} in Algorithm 6.1.

Algorithm 6.2 - Reduced System Partitioning for $Ax = b$

```

for all ( i in n ) { compute  $\lambda_{ii}$  }
for all ( i,j in N ) {
    if (  $a_{ij}\lambda_{ii}\lambda_{jj} > \alpha$  ) {  $x_i$  is lumped with  $x_j$  }
}

```

■

A simple circuit interpretation can be given for the two partitioning algorithms based on Norton equivalents[36]. Using the diagonally dominant loop criteria directly to decide whether or not to lump node x_2 with x_3 amounts to examining a circuit for which the elements to the right of x_2 and to the left of x_3 have been replaced with a current source in parallel with a 0.1 ohm resistor to ground. Using the reduced system partitioning algorithm amounts to using the exact equivalent for the circuit in Fig. 6.4, that is, to replace the elements to the right of x_2 and to the left of x_3 with their Norton equivalent, a current source in parallel with a 100.1 ohm resistor to ground. Then diagonally dominant loop test applied to this reduced system indicates that $\gamma_{\infty} \approx 0.98$, and is identical to the spectral radius computed above.

Of course, computing the inverse of A is a foolish approach to partitioning if the problem is to compute a matrix solution by relaxation. It is a useful notion though, because there are many cases where reasonable approximations to λ_i can be computed easily, as we will demonstrate in Chapter 7.

Either the diagonally dominant loop or the reduced system criteria are heuristic techniques for partitioning linear algebraic systems. The next step is to lift the technique to an approach for partitioning the differential systems of the form of Eqn. (2.2).

$$C\dot{x}(t) = Ax(t) + u(t) \quad x(0) = x_0 \quad [6.28]$$

where $C, A \in \mathbb{R}^{n \times n}$, C nonsingular, and $x(t) \in \mathbb{R}^n$. We will start by presenting the waveform equivalent of the iteration factor.

Definition 6.4: Let $x^k: [0, T] \rightarrow \mathbb{R}^n$ be the function generated by the k^{th} iteration of the WR algorithm applied to a system of the form of Eqn. (6.28). Then the WR l_∞ uniform iteration factor, γ_∞^{WR} , for the system is defined as the smallest positive number such that

$$\max_{[0, T]} \|x^{k+1}(t) - x^k(t)\|_\infty \leq \gamma_\infty^{WR} \max_{[0, T]} \|x^k(t) - x^{k-1}(t)\|_\infty$$

for any $k > 0$, any continuously differentiable initial guess v^0 , and any piecewise continuous input u . ■

There are two ways to reduce γ_∞^{WR} . The first, discussed in the Section 6.1, is to reduce the simulation interval $[0, T]$ until γ_∞^{WR} is less than one. The second approach is to partition the circuit into loosely coupled subsystems. A combination of the two techniques is needed to allow for reasonably large windows and reasonable small partitions.

As mentioned above, it is difficult to estimate γ_∞^{WR} directly for a given problem of the form of Eqn. (6.28). There are the following theorems which relate γ_∞^{WR} to iteration factors applied to a simplified system of equations.

Theorem 6.3: Let γ_∞^{WR} be the WR uniform iteration factor for a given system of equations of the form of Eqn. (6.28) solved on $[0, T]$. Then in the limit as $T \rightarrow \infty$, γ_∞^{WR} is bounded below by the spectral radius of $(L_s + D_s)^{-1}U_s$ where L_s, D_s, U_s are the strictly lower, diagonal, and strictly upper triangular portions of A given in Eqn. (6.28).

The theorem is simple to prove given the following lemma, the proof of which is given in [32].

Lemma 6.2: Let F be any linear map such that $y = Fx$, $y, x: [0, \infty) \rightarrow \mathbb{R}^n$. Define $y(s), x(s), F(s)$ as the Laplace transforms of y, x , and F respectively. Then the spectral radius of the map $F, \rho(F)$ is equal to the $\max_s \rho(F(s))$. ■

Proof of Theorem 6.3

Let L_c, D_c, U_c be the strictly lower triangular, diagonal, and upper triangular portions of C . Similarly, let L_a, D_a, U_a be the strictly lower triangular, diagonal, and upper triangular portions of A . Using this notation, the Gauss-Seidel WR iteration equation applied to solving Eqn. (6.28) is

$$(L_c + D_c)x^{k+1}(t) + U_c x^k(t) = (L_a + D_a)x^{k+1}(t) + U_a x^k(t). \quad [6.29]$$

Define $\varepsilon^k(t) = x^k(t) - x^{k-1}(t)$. Taking the difference between the $k+1^{\text{th}}$ and k^{th} iteration of Eqn. (6.29) yields

$$(L_c + D_c)\varepsilon^{k+1}(t) + U_c \varepsilon^k(t) = (L_a + D_a)\varepsilon^{k+1}(t) + U_a \varepsilon^k(t). \quad [6.30]$$

Taking the Laplace transform of Eqn. (6.30) yields,

$$s(L_c + D_c)\varepsilon^{k+1}(s) + sU_c \varepsilon^k(s) = (L_a + D_a)\varepsilon^{k+1}(s) + U_a \varepsilon^k(s). \quad [6.31]$$

Reorganizing, assuming the diagonal elements of C are nonzero,

$$\varepsilon^{k+1}(s) = [s(L_c + D_c) + (L_a + D_a)]^{-1}(sU_c + U_a)\varepsilon^k(s), \quad [6.32]$$

from which it can be seen that

$$F(s) = [s(L_c + D_c) + (L_a + D_a)]^{-1}(sU_c + U_a) \quad [6.33]$$

In particular,

$$F(0) = [L_a + D_a]^{-1}U_a$$

which, given Lemma 6.2, proves the theorem. ■

Theorem 6.4: Let γ_{\pm}^{WR} be the WR uniform iteration factor for a given system of equations of the form of Eqn. (2.1). Then γ_{\pm}^{WR} is bounded below by the spectral radius of $(L_c + D_c)^{-1}U_c$ where L_c, D_c, U_c are the strictly lower, diagonal, and strictly upper triangular portions of C given in Eqn. (6.28).

Proof of Theorem 6.4

Algebraically reorganizing Eqn. (6.30),

$$\varepsilon^{k+1}(t) = -(L_c + D_c)^{-1}U_c \varepsilon^k(t) - \quad [6.35]$$

$$(L_c + D_c)^{-1}(L_a + D_a)\varepsilon^{k+1}(t) + (L_c + D_c)^{-1}U_a \varepsilon^k(t).$$

Integrating Eqn. (6.35) and using the fact that $\varepsilon(0) = 0$,

$$\varepsilon^{k+1}(t) = -(L_c + D_c)^{-1}U_c \varepsilon^k(t) - \quad [6.36]$$

$$\int_0^t (L_c + D_c)^{-1}(L_a + D_a)\varepsilon^{k+1}(\tau) d\tau + \int_0^t (L_c + D_c)^{-1}U_a \varepsilon^k(\tau) d\tau.$$

Since Eqn. (6.36) holds for all t , it holds as $t \rightarrow 0$, which proves the theorem. ■

In Eqn. (6.28), C represents the matrix of linear capacitors, and A is the net circuit currents generated by conductances. The two theorems above indicate that it is possible to get lower bound estimates of γ_{\pm}^{WR} by examining circuits where only the capacitances and conductances are independently present. These estimates are lower bounds, hence, to decrease γ_{\pm}^{WR} below a desired α , it is *necessary* to partition in such a way that the iteration factors for the Gauss-Seidel iteration applied to the algebraic systems are decreased below α .

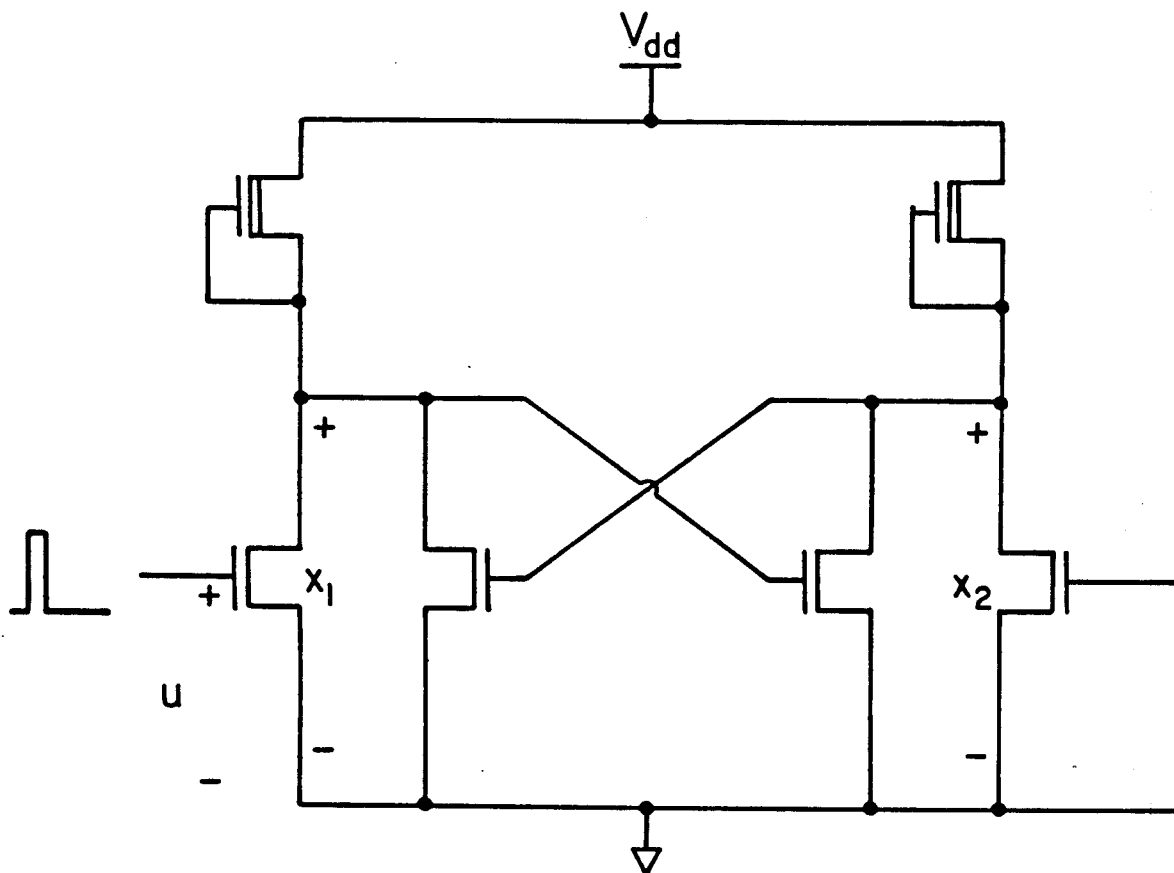


Figure 6.1a - Cross-Coupled Nor Gate

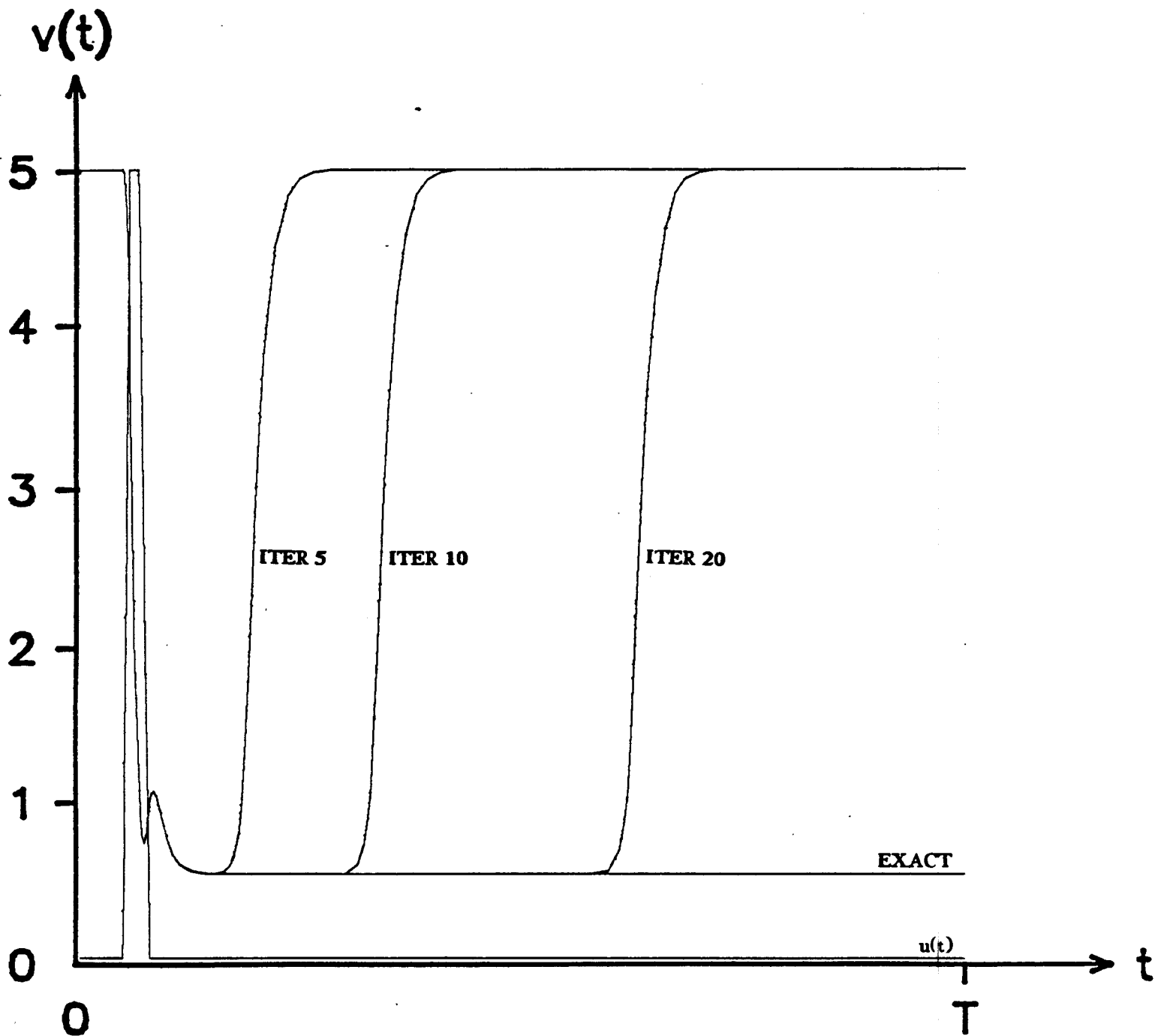


Figure 6.1b - WR Iteration from Cross-Coupled Nor Gate

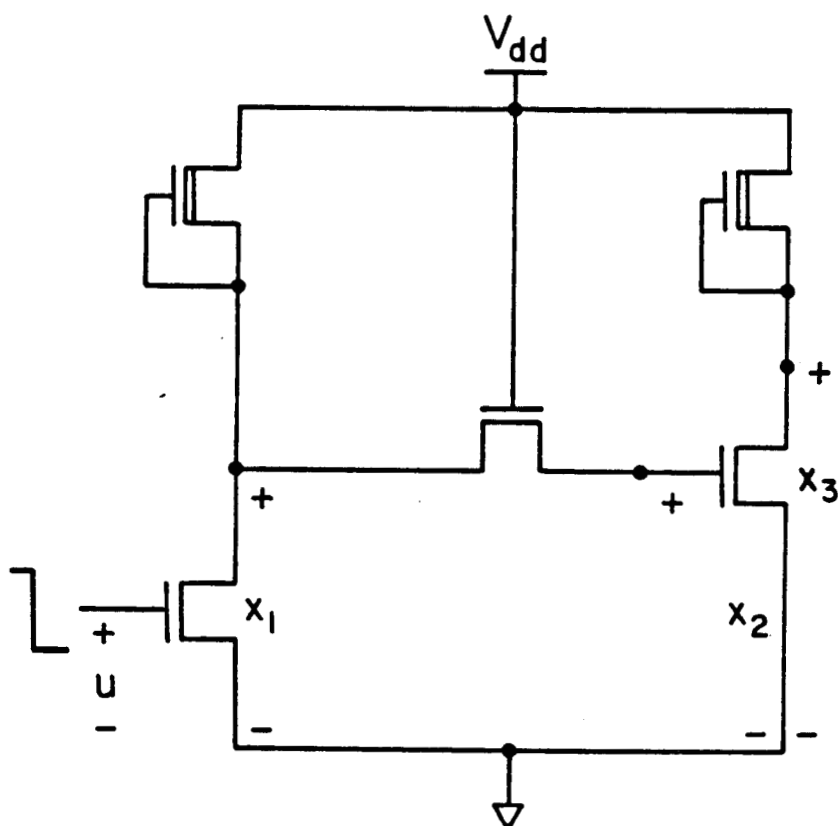


Figure 6.2a - Shift Register

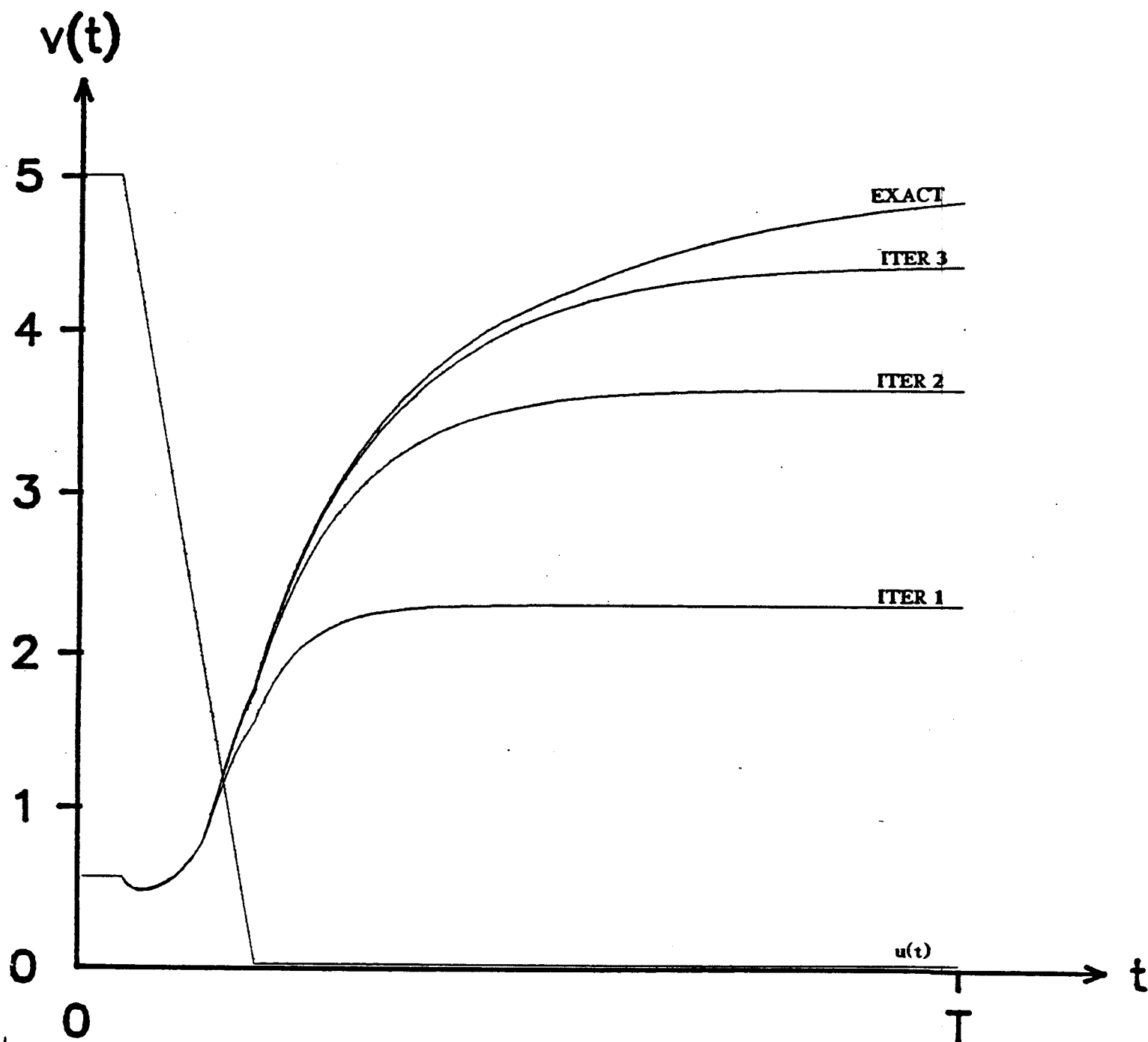


Figure 6.2b - WR Iterations from Shift Register

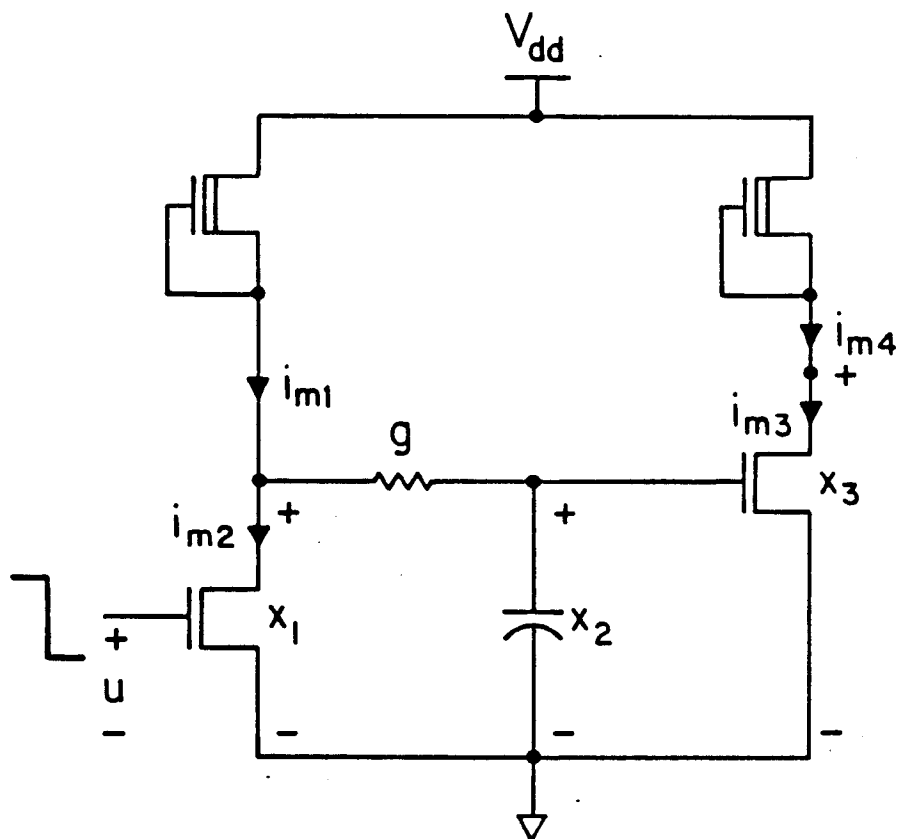


Figure 6.3a - Inverter with Delay

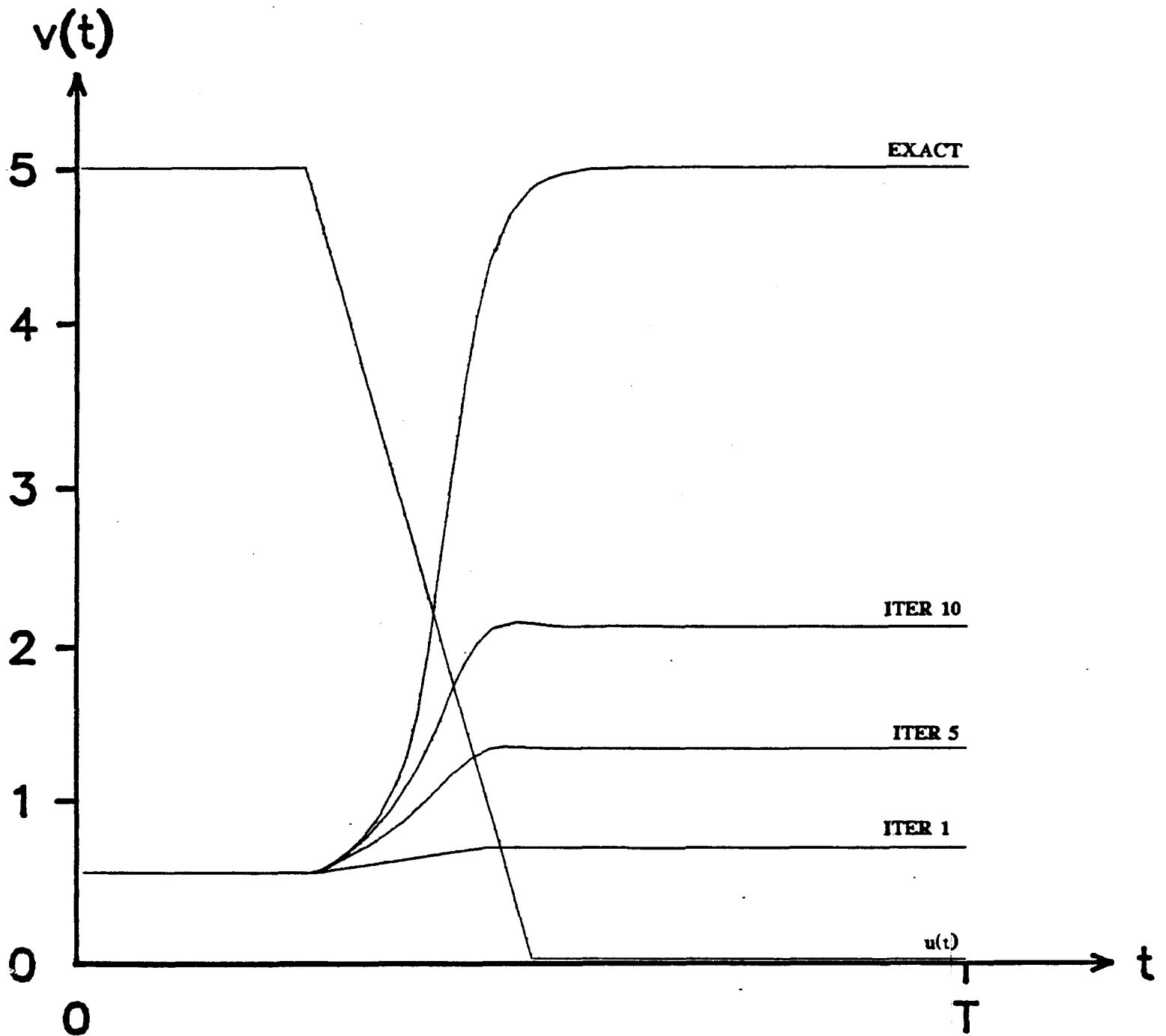


Figure 6.3b - WR Iterations from Inverter with Delay

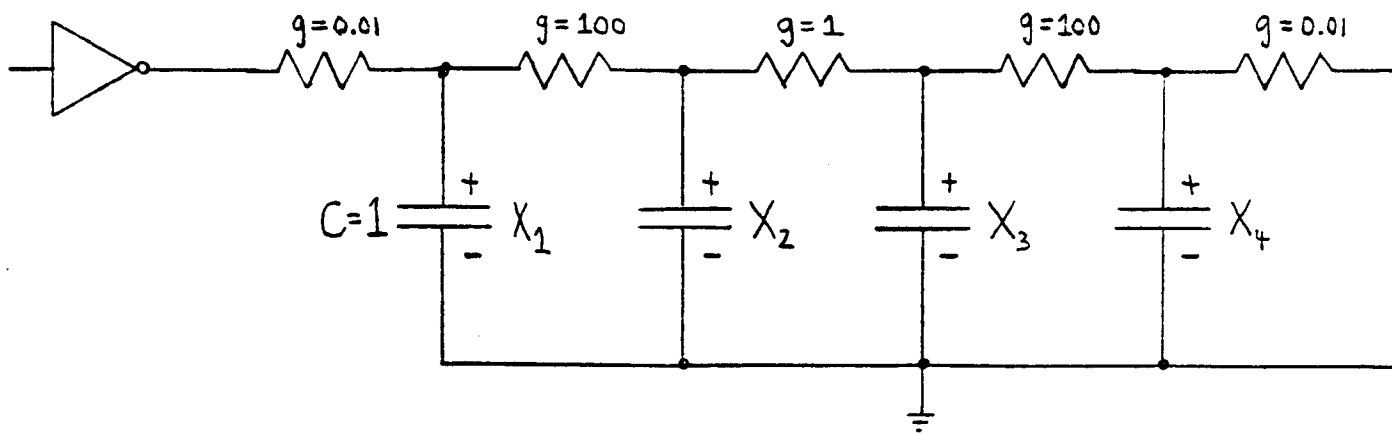


Figure 6.4 - Inverter Driving a Series of Resistors

CHAPTER 7 - THE IMPLEMENTATION OF WR IN RELAX2.3

In this Chapter, a description of the implementation of the WR algorithm in the RELAX2.3 program is given. We start with a brief overview of the steps performed in the RELAX2.3 program when simulating a circuit. A detailed description of the major steps is contained in the sections that follow.

The first step in simulating a circuit using the RELAX2.3 program is to create the circuit description file. In this file a user must specify device model parameters, circuit topology, analysis specifications, and plotting requests. The circuit topology can be described in as hierarchical or flat a form as the user desires[60]. This circuit description file is used as an input to the RELAX2.3 program, whose first step is to flatten the hierarchy.

Before applying the WR algorithm, the flattened circuit is decomposed into a collection of *subcircuits*. This is done by partitioning the circuit into clusters of tightly coupled nodes. Then the elements (e.g. transistors, resistors, capacitors) that connect to any of the nodes in a given cluster are gathered together to make the subcircuits. Once the entire circuit has been carved up into subcircuits, the subcircuits are ordered, or scheduled, starting with subcircuits that are connected to the user-defined inputs and then following the natural directionality of the circuit (as much as possible).

After a large circuit has been broken up into subcircuits, and these subcircuits have been ordered, the RELAX2.3 program begins the waveform relaxation process. An initial guess is made for each of the node voltage waveforms. Then the numerical solution for each of the subcircuits is computed in the order determined above. The computation is performed using a variable-timestep trapezoidal rule numerical integration algorithm, with local truncation error timestep control[1]. To perform the numerical integration, those nodes in the subcircuit that were not part of the cluster around which the subcircuit was built are treated as external time-varying voltage sources. The values for the external voltage sources are either the initial guess waveforms, or if the subcircuit containing the external node was simulated previously, that computed waveform. As the node waveforms are

computed, they replace the existing waveforms (initial guesses or previous iterations), and the process is repeated until the waveforms converge.

As mentioned in Chapter 6, the WR algorithm becomes inefficient when used to simulate digital circuits with logical feedback (e.g. finite state machines, ring oscillators, etc.) for many cycles, because the relaxation converges in a very nonuniform manner. For this reason the RELAX2.3 program does not actually perform the relaxation iterations by computing the transient behavior of each subcircuit for the entire user-defined simulation interval. Instead, the RELAX2.3 program uses a modified WR algorithm[17], in which the relaxation is only performed for a small piece of the user-defined simulation interval at a time. Exactly how large a piece of the waveform, referred to as a *window* to use is determined automatically, at the beginning of every WR iteration.

If the WR algorithm applied to very large circuits, it is often the case that some pieces of the circuit will converge much more rapidly than others. This phenomenon, called partial waveform convergence, can be exploited to improve the overall efficiency of the WR method. The details of the algorithm for avoiding recomputing the waveforms that have already converged are given in Section 7.5.

As a final point, in Chapter 5 it was mentioned that when the WR iteration equations are solved using a numerical integration algorithm, the resulting discretized WR algorithm is not guaranteed to converge *unless* the discretization error is driven to zero with the iterations. For this reason, the RELAX2.3 program reduces the acceptable local truncation error criteria used for selecting the numerical integration timesteps as the iterations in a given window progress.

SECTION 7.1 - PARTITIONING MOS CIRCUITS

As was shown in Section 6.2, the convergence of WR is greatly accelerated if groups of tightly coupled nodes are solved together as one subsystem or subcircuit. For this reason the RELAX2.3 program groups together tightly coupled nodes into subcircuits before beginning the relaxation process. The algorithms used in the RELAX2.3 program to partition large MOS circuits is based on Al-

gorithm 6.2 for partitioning linear algebraic systems, and Theorems 6.3 and 6.4 that relate the problem of partitioning linear algebraic systems to partitioning linear differential systems.

MOS circuits are not linear, so the ideas presented in Section 6.2 must be modified if they are to be applied to nonlinear systems. The RELAX2.3 program uses several conservative heuristics (conservative in the sense that they tend to error on the side of producing larger than optimal sub-circuits) to handle the nonlinear MOS transistors. The first heuristic is that each of the MOS transistors is initially treated as a nonlinear resistor between the transistor's source and drain, and the coupling between the gate and source and gate and drain is considered separately, during scheduling (See Section 7.2). With this simplification, the following algorithm for partitioning circuits with two-terminal linear and nonlinear resistances is applied.

Algorithm 7.1 - (Conductance Partitioning)

```

for each ( conductive element in the circuit ) {
     $g_3 \leftarrow$  maximum element conductance over all  $v$ .
    Remove the element from the circuit.
    Replace each of the other conductances in the circuit by its minimum values over all  $v$ .
    Compute  $g_1$  and  $g_2$ , the Norton Equivalent conductances at the element terminals
    If (  $\frac{g_3}{(g_2 + g_3)} \frac{g_3}{(g_1 + g_3)} > \alpha$  ) { Here,  $\alpha$  is the desired WR iteration factor, typically 0.3
        Tie the two terminal nodes together.
    }
}

```

Computing the Norton equivalent conductances, G_{eq} , at a node can be performed using a simple recursive formula if there are no loops of conductances among only non-voltage source nodes. Note that this recursion will not be very deep. The recursion will stop at any MOS transistor, because the minimum conductance of the MOS transistor is zero.

Algorithm 7.2 - (Norton Equivalent Conductance for Node i)

```

 $G_{eq} \leftarrow 0.0$ 
foreach ( conductive element incident at node  $i$  ) {
     $G \leftarrow$  element conductance
     $nodej \leftarrow$  the conductive element's other node.
}

```

```

    If ( nodej is a voltage source node ) {
         $Geq \rightarrow Geq + G$ 
    }
    else {
         $Geqj \leftarrow$  Norton equivalent conductance at nodej with this element removed.
         $Geq \leftarrow Geq + (G \times Geqj)/(G + Geqj)$ 
    }
}

```

■

If the circuit does contain conductance loops among only non-voltage source nodes, the above algorithm can still be used if the recursion is truncated in such a way that no circuit node is visited twice. In this case, only an estimate of the Norton equivalent will be computed.

The conductance partitioning algorithm is justified by Theorem 6.3, that the WR iteration factor is bounded below by the iteration factor for solving just the algebraic portion of the problem. Theorem 6.4 suggests that an analogous algorithm to Algorithm 7.1 be constructed for the capacitive elements in the circuit. Since the capacitance problem is almost identical in nature to the conductance problem, a capacitance partitioning algorithm can follow almost the same strategy as the conductance partitioning. The difference is that instead of comparing floating capacitances to Norton equivalent conductances, they are compared to equivalent capacitances. These equivalent capacitances are entirely analogous to the equivalent conductances, and can be computed using the same recursive approach as in Algorithm 7.2.

The RELAX2.3 program uses both conductance and capacitive partitioning, and forms subcircuits from the union of the two results. The algorithm has been applied to a wide variety of MOS digital circuits, including a large VHSIC memory circuit with 2900 nodes and over 3500 parasitic components. The results have always matched the best attempts at hand partitioning, in as many instances as we had the patience to check. However, it is likely that if the method is applied to larger problems, the subcircuits produced may become quite large. Should this be the case, the present simple algorithm could be extended, so that an additional pass is made over only the excessively large subcircuits, to subpartitioning them using more sophisticated algorithms. In particular, to use better

estimates of the equivalent conductances and capacitances, as the present algorithm may be unnecessarily conservative.

SECTION 7.2 - ORDERING THE SUBSYSTEM COMPUTATION

When applying the Gauss-Seidel WR algorithm to a decomposed system of differential equations, the order in which the equations are solved can strongly effect the number of WR iterations required to achieve satisfactory convergence. In order to explain this effect, consider the case where there are only grounded two-terminal capacitors for each node of the circuit. Thus, the matrix $C(x,u)$ of Eqn. (2.2) is diagonal. Then let the *dependency matrix* of f in Eqn. (2.2) be defined as a zero-one matrix $P = [p_{ij}]$ such that $p_{ij} = 1$ if f_i depends on x_j , $p_{ij} = 0$ otherwise. Note that P also represents the zero-nonzero structure of the Jacobian of f .

If P is lower triangular, then one iteration of the Gauss-Seidel WR algorithm will produce the exact solution to the original differential equation system (in practice, two iterations will be performed because a second iteration is needed to verify that convergence has been achieved). If P is not lower triangular, but the dependence of the f_i component of f on x_j , $i < j$, is "weak", then the result of one iteration of the Gauss-Seidel WR algorithm will be close to the exact solution, and subsequent iterations will converge rapidly. For this reason, when applying relaxation techniques to the solution of circuit equations, the technique can be made much more efficient by reordering the equations to make P as close to a lower triangular matrix as possible.

As discussed in Section 6.2, subsets of nodes in a large circuit may be mutually tightly coupled, and in order to insure that the relaxation algorithm converges rapidly when applied to such a circuit, these subsets are grouped together into subcircuits and solved with direct methods. This corresponds to *block* relaxation method, and an ordering algorithm applied to a system being solved with block relaxation should attempt to make f as *block lower triangular* as possible.

In some sense, partitioning and ordering the subsystem of equations are performing similar functions. They are both attempting to eliminate slow relaxation convergence due to two nodes in a

large circuit being tightly coupled. There is, however, a *key* difference. If, for example, x_i is strongly dependent on x_j and x_j is strongly dependent on x_i , then a partitioning algorithm should lump the two nodes together into one subcircuit. However, if x_i is strongly dependent on x_j , but x_j is *weakly* dependent on x_i , then node i and node j should not be lumped together, but the ordering algorithm should insure that the system is block lower triangular by ordering the equations so that x_j is computed before computing x_i .

Resistors and capacitors do not exhibit the kind of unidirectional coupling that is of concern to the ordering algorithm. In fact, the only element type of concern to the ordering algorithm are transistors, because they exhibit unidirectional coupling. That is, the drain and source terminals of an MOS transistor are strongly dependent on the gate terminal of the transistor, but the gate is almost independent of the drain and source. Clearly, this implies that the subcircuits containing the given transistor's drain or source should be analyzed after the subcircuit containing the given transistor's gate.

To devise an algorithm to carry out this task, it is convenient to introduce the *dependency graph* of the partitioned circuit. If we represent the circuit with a directed graph $G(X, E)$, where the set of nodes, X , is in one-to-one correspondence with the subcircuits obtained by a partitioner, and where there is a directed edge between the node corresponding to subcircuit i and the node corresponding to subcircuit j if there is a transistor whose gate is in subcircuit i and whose drain or source is in subcircuit j . If the graph is acyclic, it can be leveled, i.e. all the nodes can be ordered in *levels* so that a node in level i can have incoming edges only from nodes in levels lower than i . The ordering so obtained is the one used by RELAX2.3 to process the subcircuits.

However, there may be cases where cycles exist in the graph. In this case, either the subcircuit definitions are changed by grouping two or more subcircuits together, effectively performing part of the partitioning task (As alluded to in Section 7.1), or edges of the graph are discarded to remove the cycles. In either case, at the end of this process an acyclic graph and an ordering of the subcircuits corresponding to the leveling of the (perhaps altered) graph is obtained.

One question remains, which is when to repartition to remove a feedback loop versus breaking the loop. As the example Section 6.1 indicates, if signal propagation around the feedback loop is fast compared to the size of the window, the relaxation convergence will be slow and nonuniform.. For this reason, the ordering algorithm makes the decision about partitioning based on an estimate of the delay around the feedback loop. If it is smaller than one percent(somewhat arbitrarily chosen) of the simulation interval, the feedback loop is removed by repartitioning. If the delay is larger, then the feedback loop is broken by removing an edge from the directed graph.

Algorithm 7.3 - (Relax2.3 Subcircuit Ordering Algorithm)

Initialization.

ordered__list = NULL;

unordered__list = List of subcircuits from the partitioner;

Main Loop.

```
while ( unordered__list ≠ NULL ) {
    none__ordered == FALSE;
    while ( none__ordered == FALSE ) {
        none__ordered == TRUE;
        for each (subcircuit in the unordered__list) {
            if (all subcircuits on incoming arcs are on ordered__list) {
                none__ordered = FALSE;
                append__to__end__of__ordered__list(subcircuit);
                delete__from__unordered__list(subcircuit);
            }
        }
    }
    if ( unordered__list ≠ NULL ) { Must be a feedback loop.
        found__loop = FALSE;
        depth = 1;
        while ( found__loop == FALSE ) {
            depth = depth + 1;
            for each ( subcircuit in the unordered list ) {
                if ( there exists a loop of length = depth ) {
                    found__loop = TRUE;
                    if ( delay around the loop > 0.01 * the simulation interval ) {
                        break the loop
                    }
                }
                else {
                    collapse loop into one subcircuit.
                }
            }
        }
    }
}
```

SECTION 7.3 - COMPUTATION OF THE SUBSYSTEM WAVEFORMS

As in standard circuit simulators, the RELAX2.3 program solves Eqn. (4.4) using a numerical integration method with varying timesteps. Since the major aim of the RELAX2.3 program is to simulate digital circuits, the integration method was chosen based on how effectively it solves problems with the properties of digital circuits. Digital circuits are very stiff, therefore only A-stable integration methods were considered. In addition, digital circuits contain very rapid transitions, and low order one-step integration methods are usually suggested for such problems. Although the Backward-Euler method is computationally the simplest A-stable one-step method, the trapezoidal rule, an A-stable second-order one-step method, was chosen instead because of its better accuracy.

There is a second important reason for choosing the trapezoidal integration algorithm over the implicit-Euler formula. If the WR algorithm is used to solve the system, and a numerical integration method is used to solve the WR iteration equations, then the upper bound on the timestep to guarantee WR convergence (see Chapter 5) is a function of the integration method. This timestep constraint is larger for the trapezoidal rule than for implicit-Euler. To show this, consider the simple case of the WR algorithm applied to Eqn. (2.2) with $C(v,u) = I$, that is assume all the capacitors are linear, grounded and unity. The WR iteration equations become

$$\dot{v}^{k+1} = \dot{v}^k + f'(v^{k+1}, v^k, u). \quad [7.1]$$

where f' is as defined in Section 4.2. Now consider computing the first time step of the implicit-Euler discretized WR algorithm:

$$v^{k+1}(h) - v_0 = (v^k(h) - v_0) + f'(v^{k+1}(n+1), v^k(n+1), u). \quad [7.2]$$

Applying the trapezoidal rule yields:

$$v^{k+1}(h) - v_0 = (v^k(h) - v_0) + 0.5h f'(v^{k+1}(n+1), v^k(n+1), u) + 0.5h f(v_0, u). \quad [7.3]$$

The reason for the relaxation iteration is to resolve $f'(v^{k+1}(n+1), v^k(n+1), u)$, and it plays a smaller role in Eqn. (7.3) than in Eqn. (7.2), and therefore the iteration of Eqn. (7.3) will achieve and given convergence threshold faster.

Given a timestep h , the trapezoidal integration method applied to Eqn. (4.4) yields:

$$q(t+h) - q(t) - 0.5h(\hat{f}(q(t+h), u) + \hat{f}(q(t), u)) = 0 \quad [7.4]$$

The above equation is a nonlinear algebraic equation in q . The user is usually more interested in the voltage, so before solving Eqn. (7.4) we substitute for q in terms of v .

$$q(v(t+h)) - q(v(t)) - 0.5h[f(v(t+h), u) + f(v(t), u)] = 0 \quad [7.5]$$

In Eqn. (7.5) $v(t)$ and $q(t)$ are known, and the equation must be solved to compute $v(t+h)$. Nonlinear algebraic systems generated by integration methods are usually solved using the iterative Newton-Raphson method. This is because Newton methods have quadratic convergence properties and because they are guaranteed to converge if the initial guess is close enough to the correct solution. The general Newton-Raphson iteration equation to solve $F(x) = 0$ is

$$J_F(x^k)(x^k - x^{k-1}) = -F(x^{k-1}) \quad [7.6]$$

where J_F is the jacobian of F with respect to x . The iteration is continued until $\|x^k - x^{k-1}\| < \epsilon$ and $F(x)$ is close enough to 0. If the Newton algorithm is used to solve Eqn. (7.5) for $v(t+h)$, the residue, $F(v^k(t+h))$, is:

$$F(v^k(t+h)) = q(v^k(t+h)) - q(v(t)) - 0.5h(f(v^k(t+h), u) + f(v(t), u)) \quad [7.7]$$

and the Jacobian of $F(v^k(t+h))$, $J_F(v^k(t+h))$ is:

$$J_F(v^k(t+h)) = C(v^k(t+h), u) + 0.5h \frac{\partial f(v^k(t+h), u)}{\partial v}.$$

Then $v^{k+1}(t+h)$ is derived from $v^k(t+h)$ by solving the linear system of equations:

$$J_F(v^k(t+h)) [v^{k+1}(t+h) - v^k(t+h)] = -F(v^k(t+h)) \quad [7.8]$$

The Newton iteration is continued until sufficient convergence is achieved, that is $\|v^{k+1}(t+h) - v^k(t+h)\| < \epsilon$ and $F(v^k(t+h))$ is close enough to zero.

Each iteration of the Newton algorithm requires a function evaluation, a Jacobian evaluation, and a matrix solution. For the algebraic systems generated by the numerical integration of MOS digital circuits it is often inefficient to evaluate the Jacobian every Newton iteration. If the Jacobian is reevaluated only every few Newton iterations[27], the number of iterations required to achieve convergence is usually unchanged and the computation required is significantly reduced. Not only are Jacobian evaluations skipped, but if the matrix solution is computed by LU factorization[40], subsequent matrix solutions using the same matrix can skip the LU factorization step. In the RELAX2.3 program the Jacobian is evaluated every third iteration, this choice based on experimental evidence in several examples given in the table below.

TABLE 7.1 - CPU TIME VS # OF NEWTON ITERATIONS/JACOBIAN EVALUATION					
Circuit	Devices	1	2	3	4
Ring Osc.	7	0.95s	0.77s	0.71s	0.75s
Oper. Amp	25	6.28s	5.2s	4.52s	4.67s
flip-flop	33	20.47s	16.82s	13.93s	13.67s
Cmos Memory	621	1080s	976s	885s	886s

On Vax11/780 running Unix

The integration method used in the SPICE2 program is very similar to the direct method used in RELAX2.3. Both use the trapezoidal integration formula with local truncation error timestep control, the Newton method to solve the algebraic system, and sparse LU factorization to perform the matrix solution. However, as can be seen from Table 7.2, the RELAX2.3 program, using the direct

method described above, is eight to twenty times faster than the SPICE2 program. This can be attributed to many factors. The first is that RELAX2.3 is written in "C", SPICE2 is in FORTRAN, and "C" programs under the UNIX operating system run almost a factor of two times faster than FORTRAN programs. The other factor of four to ten is due to more sophisticated programming techniques, the more efficient equation formulation and the modified Newton method mentioned above, and better numerical integration error control.

TABLE 7.2 - RELAX2.3 (DIRECT) VS SPICE ON INDUSTRIAL CIRCUITS				
Circuit	Devices	SPICE2	RELAX2.3	Ratio
Ring Osc.	7	17s	0.75s	22
Op-amp	25	42s	5s	8
uP Control	232	1400s	90s	15
Cmos Memory	621	10400s	995s	10
4-bit Counter	259	4300s	540s	8
Encode-Decode	1326	115,840s	5000s	23

On Vax11/780 running Unix

It should be pointed out that without a fundamentally new circuit simulation method, just by carefully exploiting some very general properties of MOS digital circuits, almost an order of magnitude decrease in computation time has been achieved over the much more general SPICE2 program.

SECTION 7.4 - WINDOWSIZE DETERMINATION

As mentioned in Section 6.2, the WR algorithm used in RELAX2.3 becomes inefficient when used to simulate digital circuits with logical feedback(e.g. finite state machines, ring oscillators, etc.) for many cycles. However, the WR algorithm can still be very efficient if the relaxation is only performed on a piece of the waveform to be computed at a time. For general circuits, an ideal situation would be to break the simulation interval into windows over which every time point of the iteration waveform moves closer to the correct solution. However, if the windows are too small some of the advantages of waveform relaxation are lost. One cannot take advantage of a digital circuit's natural latency over the entire waveform, but only in that window; the scheduling overhead increases when

the windows become smaller, as each circuit lump must be scheduled once for each window; and if the windows are made very small, timesteps chosen to calculate the waveforms will be limited by the window size rather than by the discretization error, and unnecessary calculations will be performed.

Rather than use a conservative *a priori* lower bound as given in Theorem 6.2, in the RELAX2.3 program, the "window size" is determined dynamically, by two criteria. The first criterion is to pick the window size to limit the number of timepoints required to represent each node waveform in a window. This puts a strict upper bound on the amount of storage needed for the waveforms, and thus allows the RELAX2.3 program to avoid dynamically managing waveform storage space. The second criterion is to try to pick the window size so that the convergence of the WR is rapid, in particular, that the waveforms approach the correct solution in a uniform manner over the entire window. The RELAX2.3 program presently uses the following window size determination algorithm:

Algorithm 7.4 (RELAX2.3 Windowing Algorithm)

```

starttime = Beginning of the window
stoptime = End of the window
endtime = End of user-defined simulation interval
usedpts = Max. # of points used in the last window
maxpts = Max. # of points in a waveform buffer
prevwindow = Size of the window used in the previous iteration
if ( Not entirely converged in this window ) then {
    if ( usedpts ≥ maxpts ) then {
        Shorten window if the waveforms overran storage buffers.
        stoptime = starttime + (prevwindow * maxpts * 0.7)/usedpts;
    }
    else if ( (numiters mod 5) == 0 ) then { Half window size every five WR iterations.
        stoptime = prevwindow/2 + starttime;
    }
    else { Else just do the same window again.
        stoptime = starttime + prevwindow;
    }
}
else {
    starttime = stoptime;
    stoptime = starttime + (prevwindow * maxpts * 0.7)/usedpts;
}

```

■

At present, one twentieth of the simulation interval is being used as an initial guess for the window size. Adding a simple critical path analyzer to RELAX2.3 is being considered to provide a better initial guess.

SECTION 7.5 - PARTIAL WAVEFORM CONVERGENCE

If the WR algorithm is used to compute the time domain behavior for very large circuits, it is often the case that some pieces of the circuit will converge much more rapidly than others. The overall efficiency of the WR method can be improved if the waveforms that have already converged are not recomputed every subsequent iteration.

To take advantage of partial waveform convergence requires a simple modification to Algorithm 4.1. Before giving the exact algorithm we present the following useful definition.

Definition 7.1: Let

$$\sum_{j=1}^n C_{ij}(v(t), u(t)) v_j(t) = f_i(v(t), u(t)) \quad v^i(0) = v_{i0} \quad [7.9]$$

be the i^{th} equation of the system in Eqn. (2.2). We say $v_j(t)$ is an input to this equation if there exists some $\alpha, t \in \mathbb{R}$ and $z, y \in \mathbb{R}^n$ such that $\sum_{j=1}^n C_{ij}(z, u(t)) y_j \neq \sum_{j=1}^n C_{ij}(z + \alpha e_j, u(t)) y_j$ or $f_i(z, u(t)) \neq f_i(z + \alpha e_j, u(t))$, where e_j is the j^{th} unit vector. The input set for the i^{th} equation is the set of $j \in [1, \dots, n]$ such that $v_j(t)$ is an input ■.

The WR algorithm is then modified slightly using this notion of the set of inputs to a given ODE.

Algorithm 7.5 - WR Algorithm with Partial Waveform Convergence

The superscript k denotes the iteration count, the subscript i denotes the component index of a vector and ϵ is a small positive number. $k \leftarrow 0$

guess waveform $x^0(t)$; $t \in [0, T]$ such that $x^0(0) = x_0$

(for example, set $x^0(t) = x_0, t \in [0, T]$)

repeat {

$k \leftarrow k + 1$

foreach (i in N) {

Partialflag = TRUE

```

if (  $k = 1$  ) Partialflag = FALSE
For each (  $j < i, j \in$  input set of  $v_i$  )
    if (  $\max_{[0,T]} |v_j^k - v_j^{k-1}| > \epsilon$  ) Partialflag = FALSE
For each (  $j \geq i, j \in$  input set of  $v_i$  )
    if (  $\max_{[0,T]} |v_j^{k-1} - v_j^{k-2}| > \epsilon$  ) Partialflag = FALSE
if ( Partialflag = TRUE )  $v_i^{k+1} = v_j^k$ 
else solve

```

$$\sum_{j=1}^i C_{ij}(v_1^k, \dots, v_i^k, v_{i+1}^{k-1}, \dots, v_n^{k-1}, u) \dot{v}_j^k +$$

$$\sum_{j=i+1}^n C_{ij}(v_1^k, \dots, v_i^k, v_{i+1}^{k-1}, \dots, v_n^{k-1}, u) \dot{v}_j^{k-1} +$$

$$f_i(v_1^k, \dots, v_i^k, v_{i+1}^{k-1}, \dots, v_n^{k-1}, u) = 0$$

for ($v_i^k(t); t \in [0, T]$), with the initial condition $v_i^k(0) = v_{i0}$

```

}
} until (  $\max_{1 \leq i \leq n} \max_{t \in [0, T]} |v_i^k(t) - v_i^{k-1}(t)| \leq \epsilon$  )
      that is, until the iteration converges.

```

■

SECTION 7.6 - EXPERIMENTAL RESULTS

The degree to which the WR algorithm improves circuit simulation efficiency can be traced to two properties of a circuit. The first, mentioned before, is the differences in the rates of change of voltages in the system, as this will determine how much efficiency is gained by solving the subsystems with independent integration timesteps. The second is the amount of coupling between the subsystems. If the subsystems are tightly coupled, then many relaxation iterations will be required to achieve convergence, and the advantage gained by solving each subsystem with its own timestep will be lost. To show this interaction for a practical example, we will use the Relax2.3[13] program to compare the computation time required to simulate a 141-node CMOS memory circuit using standard direct methods and using the WR algorithm. In order to demonstrate the effect of tighter coupling, the CMOS memory circuit will be simulated using several values of a parameter XQC, which is the percent of the gate oxide capacitance that is considered as gate-drain or gate-source overlap capacitance.

TABLE 7.3 - DIRECT VS WR ON A MEMORY CIRCUIT WITH DIFFERENT COUPLINGS				
METHOD	XQC	TIMEPOINTS	# WR ITERS	CPU TIME
Direct	0.01	124,539	1	933s
WR	0.01	17,728	2.5	304s
Direct	0.05	122,988	1	945s
WR	0.05	19,199	3	410s
Direct	0.2	118,335	1	917s
WR	0.2	19,193	4	530s
Direct	0.33	115,233	1	895s
WR	0.33	19,315	6.5	707s

The results in Table 7.3 are exactly as expected. As the coupling increases, the number of WR iterations required increases, and the difference in the simulation time for WR and the direct method decreases.

It is possible to verify, for this example, our claim of the nature of the efficiencies of using WR. Consider the number of timepoints computed by the direct method versus the number of computed timepoints for the WR method in the final iteration. By comparing these two numbers, a bound can be put on the maximum speed increase that can be achieved by solving different subsystems using different timesteps (Note that we are only considering the number of timepoints computed by the WR method in the final iteration, because we are only interested in the number of timepoints needed to accurately represent the given waveform).

The total number of timepoints computed for each of the simulation cases of the memory circuit example is also given in Table 7.3. This number is the sum of the computed timepoints over all the waveforms in the circuit. If most of the efficiency of a decomposition method stems from solving each of the subsystems with its own timestep, then the maximum improvement that could be gained from a decomposition integration method would be the ratio of the number of timepoints computed using the direct method compared to the number of timepoints computed in the final WR iteration. As can be seen from the Table 7.3, for the CMOS memory example this ratio is approximately six. In order to compute the actual efficiency of the WR method, the average number of WR iterations

performed must be considered, because for each WR iteration, the set of timepoints is recomputed. Then, if our claims above are correct, when the ratio of the number of timepoints for the direct method to the number of WR timepoints is divided into the average number of relaxation iterations, the result should be almost equal to the ratio of WR computation time to direct computation time. And as Table 7.3 shows, it is.

In the above analysis we have ignored an important advantage of relaxation methods: that they avoid large matrix solutions. This is a reasonable assumption for the above example because the matrix operations account for only a small percentage of the computations, even when direct methods are used. However, for much larger problems, of the order of several thousand nodes, the time to perform the large matrix solutions required by direct methods will dominate. In those cases WR methods should compare even more favorably because they avoid these large matrix solutions.

Finally, in Table 7.4, we present several circuits that have been simulated using RELAX2.3 with direct and WR methods.

TABLE 7.4 - DIRECT METHODS VS WR FOR SEVERAL INDUSTRIAL CIRCUITS			
Circuit	Devices	DIRECT	WR
uP Control	232	90s*	45s*
Cmos Memory	621	995s*	308s*
4-bit Counter	259	540s*	299s*
Inverter Chain	250	98s**	38s**
Digital Filter	1082	1800s*	520s*
Encode-Decode	3295	5000s*	1500s*
VHSIC Memory	625	17174s**	12505s**

*On Vax11/780 running Unix using Shichman-Hodges Mosfet model

**On Vax11/780 running VMS using Yang-Chatterjee Mosfet model

CHAPTER 8 - PARALLEL WR ALGORITHMS

Exploiting parallel computation for circuit simulation is extremely important because the size of the circuits for which circuit simulation has been applied has grown at rate that far exceeds the increase in computational power due to technological improvement. The *only* way to keep pace with the increasing demand is to be able to apply many processors to the problem, and the number of processors that can be used must scale up with the size of the problem.

A variety of techniques for the parallel solution of ordinary differential equations have been examined in the literature[63]. For circuit simulation, four techniques have been applied. The SPICE2 program was rewritten to take advantage of the Cray Computer vector capability[48]; a parallel version of a similar direct method has been implemented on the Cosmic-Cube, a message-passing based parallel computer; the Gauss-Jacobi form of the algebraic relaxation-Newton algorithm presented in Section 3.2 has been implemented on both a shared-memory computer, the Sequent Balance 3000[64], and ITM's Connection Machine[65]; and a version of the Iterated Timing Analysis algorithm (Section 3.2) has been implemented on the BBN Butterfly[34].

In this chapter, the implementation of two WR-based parallel circuit simulation algorithms on a shared memory computer will be described. We will start by presenting a brief overview of the aspects of a shared-memory computer that effect the algorithm implementation, and then describe the two parallel WR algorithms, one based on using a mixture of Gauss-Seidel and Gauss-Jacobi relaxation, and the other based on pipelining the waveform computation. For each algorithm, experimental results will be presented.

SECTION 8.1 - A BRIEF OVERVIEW OF THE SHARED MEMORY COMPUTER

When attempting to write efficient programs for serial computers, knowledge about the specific details of the architecture is useful, but not essential. This is not the case for programming on a parallel computer. Specific details about the architecture can influence decisions about the implemen-

tation of an algorithm, and can even effect the choice of algorithm. Since the algorithms that will be described below were implemented on the Sequent Balance 8000, a shared-memory parallel computer, in this section we will describe those aspects of the architecture that effected the implementation of parallel versions of the WR algorithm. For a more detailed treatment of this subject, see[56].

The key problem in designing a parallel processor is that of communication between the processors. One simple approach is to design a parallel computer by gathering together many standard serial computers, and connecting them together with a communication network. Usually such computers are referred to as *message-passing* parallel computers, because data is tranferred between the many processors by passing messages on the communication network. The disadvantage of such a system is that in order to move data from the memory of one processor into the memory of the second processor, both the transmitting and receiving processors must be involved.

Another approach to the problem of communicating between parallel processors is to redesign the memory system, so that the aggregate memory of all the processors is directly addressable by any one of the individual processors. Such a system is referred to as a *shared-memory* system because the processors are all sharing the single resource, the memory. The main advantages of a shared-memory machine is that it is not necessary to explicitly transfer data from one processor to another. When a processor needs data from another processor, it simply reads from the memory locations in which the other processor has written. This also allows for more dynamic algorithm structures, because it is not necessary to determine beforehand which processors will need the results of a given calculation. The disadvantages of the shared-memory computer are that all processors must contend for a single resource, the memory, and guaranteed synchronization between processors is not simple without special-purpose hardware.

One of the most important aspects of a shared-memory parallel computer is how the memory is distributed among the individual processors. There are fundamentally only two choices, either each processor has a portion of the shared memory which it can access rapidly, and that others can access

but not as quickly, or all the memory is centralized, and the many processors contend on an equal footing for access to it.

If the memory is distributed among the processors, then a parallel algorithm will perform better if the data for the computation can be partitioned so that each processor performs computations using only the data in its own portion of the shared memory. It is usually the case that by partitioning the data, so that each of the processors can only work on an exclusive portion of a large problem, some of the parallelism of a given algorithm will no longer be exploitable and parallel efficiency will be lost. For example, if at a certain point in the process of solving a large problem, several calculations that could be performed concurrently all require data from the same partition, those calculations will be performed serially. If simultaneously, there are no calculations to be performed using data from another partition, a processor will be idle.

A way of eliminating the loss of parallelism at the cost of complicating the control structure of the program is to have each processor use a priority scheme. In such a scheme, each processor attempts to perform calculations using data in its own partition, and then if there are none to be performed, the processor will attempt to perform calculations using data from other partitions.

Clearly, when using a shared-memory computer with distributed memory, the trade-off's of faster memory access, loss of parallelism, and more complicated control structure must be examined carefully(For an example for the case of circuit simulation see[34]).

The memory on the parallel computer used for parallel WR experiments is centralized, where all the processors contend for one large shared memory. For such an architecture, there is no advantage to partitioning the data for a large problem among the processors, as they will still have to contend for the same centralized memory pool. For this reason, the algorithms presented below ignore the issue of partitioning the data among many processors.

In order to avoid the obvious bottleneck created by having many processors contend for data out of the same central memory, most implementations of shared-memory computers that use centralized memory attempt to reduce this contention by including a large cache memory with each of

the processors. As with any cache memory scheme, these caches attempt to exploit locality of reference, that it will usually be the case that each of the processors are actively using only a small amount of data. Since this data will probably be available from the cache, for most memory accesses it will not be necessary to generate a request to the main central memory.

Using caches on a parallel computer is not as straight-forward as on a serial computer. Since there are many caches, and they are all supposed to contain a copy of the data in the central memory, and any processor can write in any memory location, it is possible for the caches to loose consistency. By this it is meant that the contents of a cache may not reflect the current contents of the central memory. For example, if the contents of memory location *A* is in both the cache for processor 1 and the cache for processor 2, and processor 1 updates *A*, then the data in the cache for processor 2 will be incorrect.

As the example demonstrates, even if the central memory is updated whenever a processor updates a location contained in its cache, a cache inconsistency can occur within a cache of another processor. There are a variety of schemes for avoiding this problem[62], but we will only mention the technique applied in the computer used for experimentation. The scheme is simple, all the caches monitor all the writes to central memory from any of the processors. If a cache contains a location being written to by any of the other processors, it updates its own copy of the data in the given location. By snooping in on the writes to central memory, each cache assures that it has the most current data.

The snooping cache consistency strategy has a particularly useful implication. It is frequently necessary to have one processor wait for another processor to finish a computation. If the computing processor is to change a location in memory when finished, the second processor can continuously test that location to determine when the computing processor is finished. Normally, this is a poor approach for a parallel environment, because the waiting processor will be continuously reading from the central memory and generating excess memory traffic. If many processors are waiting for the completion of one processor's computation, this excess traffic can become enormous, and slow the

computing processor which will have to contend with the excess traffic. If the cache architecture described above is used, the excess traffic is eliminated. Each waiting processor will keep rereading a location which will be in its own cache, and will therefore not be generating *any* central memory traffic. When the computing processor finishes, each of the other processor caches will spot the write to the monitored location in central memory and each cache will update its own copy of the data. The waiting processors will therefore be made immediately aware of the completion of the computing processor, but will not have impeded the progress of the computing processor by generating excess memory traffic.

The last aspect of the parallel computer architecture that we will consider is that of mutual exclusion or *locking*. In almost all parallel programs there are critical sections that must be performed serially, that is, only one processor should be executing the section at a time. The usual mechanism for insuring this is the *test-and-set* instruction. If a processor executes a *test-and-set* instruction on a given location in memory, the contents of the location is returned to the processor and simultaneously, if the location was not set, it is set.

The mechanism can be used to perform locking as follows. A particular location in memory is used as the lock. If a processor is about to execute a critical section of a parallel program, it first executes a test-and-set on the lock location. If the result indicates that the location was not set, then the processor can safely execute the critical section, and clear the lock location when finished. If the result indicates that the lock was already set, the processor must wait until the lock becomes clear and then retry the test-and-set.

SECTION 8.2 - A MIXED GAUSS-SEIDEL/JACOBI PARALLEL WR ALGORITHM

An obvious way of parallelizing WR is to use the Gauss-Jacobi version of WR. In this algorithm, the relaxation makes use of the waveforms computed at the previous iteration for all the subsystems. In this case, all the subsystems can be analyzed independently by different processors. One of the difficulties in applying this algorithm is that MOS digital circuits are highly directional, and, as

mentioned in Section 7.2, if this directionality is not exploited slow convergence may result. For example, consider applying WR to compute the transient response of a chain of inverters. If the first inverter's output is computed first, and the result is used to compute the second inverter's output, which is then used for the third inverter, etc., the resulting waveforms for this first iteration of the WR algorithm will be very close to the correct solution. However, if the second and third inverter outputs are computed in parallel with the first inverter's output, the results will not be close to the correct solution because no reasonable guess for the second and third inverter inputs will be available. For this reason, after partitioning, the RELAX2.3 program orders the subcircuits so that the directionality of the circuit is followed as closely as possible.

Following a strict ordering of the relaxation computation (Gauss-Seidel) does not allow for computing entire waveforms in parallel, and computing the next iteration waveforms for every subcircuit at once (Gauss-Jacobi) allows for substantial parallelism, but is not very efficient (converges more slowly). In order to preserve the efficiency of the Gauss-Seidel algorithm and allow for some of the parallelism of Gauss-Jacobi, a mixed approach can be employed. The mixed approach is based on the observation that large digital circuits contain many subsystems that can be computed in parallel without slowing the convergence. This is because large digital circuits tend to be wide. Rather than being like a long chain of gates, they are like many parallel chains, with some interaction between the chains. For this reason, it is possible to order the computation so that subcircuits in parallel "chains" can be computed in parallel, but the serial dependence inside a chain is preserved. This will not allow for as much parallelism as the Gauss-Jacobi scheme, but should preserve most of the efficiency of the Gauss-Seidel scheme.

In Algorithm 8.1, we present a probabilistic approach to attempting to follow the ordering of the subcircuits. The algorithm is set up by establishing both the space in shared memory for storage of the iteration waveforms, and a buffer or queue with the list of subcircuits in the order derived from Algorithm 7.3. Each of the processors then begins by taking a subcircuit from the queue and then computing the subcircuits output waveforms using the newest available external waveforms. When

mentioned above, this exploits the nature of the cache consistency strategy. Finally, in this case it is not necessary to separately control access to the waveforms. Since the waveforms will only be written as a result of the computations performed on their associated subcircuits, and a waveform is associated with only one subcircuit (This would not be the case if an overlapped relaxation algorithm were used) the mutual exclusion of the subcircuit queue will prevent waveform writes from colliding.

SECTION 8.3 - TIMEPOINT-PIPELINING WR ALGORITHM

It is possible to parallelize the WR algorithm while still preserving a strict ordering of the computation of the subcircuit waveforms (Gauss-Seidel), by pipelining the waveform computation. In this approach, one processor starts computing the transient response for a subcircuit. Once a first timepoint is generated, a second processor begins computing the first timepoint for the second subcircuit, while the first processor computes the second timepoint for the first subcircuit. On the next step a third processor is introduced, to compute the first timepoint for the third subcircuit, and so on.

Conceptually, the operations of a given processor in a parallel timepoint pipelining algorithm are quite simple. The algorithm is set up by establishing both the space in shared memory for storage of the iteration waveforms, and a buffer or queue with the list of subcircuits. Each of the processors then starts by taking a subcircuit from the queue. The individual processors examine their respective subcircuit's external waveforms to see if the waveform values needed to compute the next integration timestep are available. If so, the next timestep for the subcircuit is computed. Otherwise, the subcircuit is returned to the queue and the processor tries again with another subcircuit from the queue. As timepoints are computed, more of the subcircuits will have the information needed to compute their own timepoints.

As one might expect, a practical timepoint pipelining algorithm is more complicated than the conceptual algorithm. Perhaps the most obvious difficulty is that there is a tremendous overhead in having every processor search through all the subcircuits to find one of the few for which a timepoint

the waveform computation is completed, the subcircuit is temporarily discarded and the processor takes a next subcircuit off the queue. This continues until the queue is exhausted and all the processors are finished. Then queue is reset, and the processors all start picking up subcircuits again.

This algorithm is probabilistic in the sense that there is no guarantee that the transient computation for a given subcircuit will be finished before its output is needed by another subcircuit who is strongly serially dependent on the first subcircuit's output. It is likely that the given subcircuit's output will have been computed if the circuit is very wide (there are a large number of parallel chains) compared to the number of processors. In addition, since all the subcircuit outputs must be computed before any subcircuit's output is recomputed, no subcircuit will be more than one iteration behind.

Algorithm 8.1 - (Jacobi/Seidel based Parallel WR)

Initialization. Both subcircuits and waveforms in shared-memory.

queue = ordered_list_of_subcircuits

while (all_converged == FALSE) { *Parallel iteration loop. All processors execute.*

 if (processor == 1) {

 reset_queue()

 idle_count = 0

 }

 while (idle_count ≠ number of processors) {

 while (test-and-set(queuelock) == set) { *Tight loop waiting for queue to unlock. }*

Queue is locked, get next subcircuit

 NextSub = Get_next_queue_entry()

 if (NextSub == NULL) {

 increment(idle_count)

 clear(queuelock)

 }

 else { *There is another subcircuit on the queue.*

 clear(queuelock)

 Compute_Subcircuit_Waveforms(NextSub)

 Check_Waveform_Convergence(NextSub)

 }

 }

}

■

Note that the attributes of the parallel architecture have been considered in Algorithm 8.1. Since the machine is a centralized shared-memory machine, the data describing the subcircuits and the computed waveforms are left in shared memory, to be accessed as needed. Also note that each of the processors waits for the queue to be free by examining the lock variable in a tight loop. As

can be computed. It is possible to reduce the number of candidate subcircuits a processor must search by only considering those subcircuits for which at least one of the external waveforms has more timepoints than it had when the subcircuit was last processed. Clearly, this will avoid having the processors continuously rechecking subcircuits for which no new information is available, and therefore no new timestep could be computed.

This kind of selective search algorithm can be implemented by altering the way the queue of subcircuits is used. When a processor discerns that it is not possible to compute a new timepoint for a subcircuit, instead of returning the subcircuit to the queue, the subcircuit is temporarily discarded. If a processor succeeds in computing a timepoint for a subcircuit, those subcircuits that are connected to the given subcircuit, referred to as the *fanouts* of the subcircuit, are added to the queue (Of course, any of the fanouts that are already on the queue are not duplicated). In this way, the only subcircuits that will be on the queue are those for which it is likely that the waveform values needed to compute a next timepoint will be available.

Another aspect of the timepoint pipelining algorithm that increases the exploitable parallelism at the cost of slightly complicating the algorithm is to allow the timepoint pipelining to extend across iteration boundaries. For example, consider a chain of two inverters, and assume that it takes two timesteps to compute each of the inverter outputs. As before, the second timestep of the first inverter can be computed in parallel with the computation of the first timestep of the second inverter. Then, while the second timestep of the the second inverter is being computed, there is enough information to compute the first timestep of the first inverter for the *second* WR iteration.

This enhancement doesn't really complicate the conceptual algorithm, until one considers the question of when to stop. For a long chain of inverters, allowing the pipelining to extend across iteration boundaries can easily allow for the first inverter to be many iterations ahead of the last inverter. Since WR convergence can only be determined when all the waveforms for a given iteration have been computed, it may well be that the WR iteration being computed for the first inverter is many iterations beyond what is necessary to achieve satisfactory convergence. The difficulty is that

this fact will not be discovered until much later, when all inverter outputs have been computed for the iteration for which satisfactory convergence is achieved.

This is not a disastrous problem, the algorithm will still produce correct solutions, but unnecessary computations will be performed and efficiency will be degraded. The unnecessary computations are reasonably simple to avoid, by not allowing any subcircuit to start on iteration $N+1$ until nonconvergence of some waveform of iteration N has been detected. It is, of course, important to discover as quickly as possible if it will be necessary to compute iteration $N+1$, so that the pipelining of that iteration can begin. For this reason, in the timepoint pipelining algorithm presented below, convergence is checked on a timepoint by timepoint basis, immediately after a timepoint is computed.

Algorithm 8.2 - (Timepoint Pipelining WR Algorithm)

Initialization. Both subcircuits and waveforms in shared-memory.

queue = ordered_list_of_subcircuits

idle_count = 0

Max_iter_so_far is the iter after the last one for which nonconvergence was detected

max_iter_so_far = 1

Parallel iteration loop. All processors execute.

while (idle_count \neq number of processors) { *at least one processor is still computing.*

while (test-and-set(queuelock) == set) { *Tight loop waiting for queue to unlock. }*

Queue is locked, get next subcircuit in the queue for which the work that might be performed on it is for an iteration that is no more than one beyond the maximum iteration for which nonconvergence has been detected.

NextSub = Get_next_queue_entry(max_iter_so_far)

if (NextSub == NULL) {

increment(idle_count)

clear(queuelock)

}

else {

There is a subckt on the queue whose iteration is not beyond max_iter_so_far.

clear(queuelock)

Compute as many timepoints as possible with available waveform values.

repeat {

Check to see if external values needed to compute the next timestep are available.

cando = Check_for_next_step(NextSub)

if (cando == TRUE) {

Compute_Next_Step(NextSub)

converged = Check_Step_Convergence(NextSub)

if ((converged == FALSE) and (NextSub.iter_count == max_iter_so_far)) {

Keep max_iter_so_far ahead of the nonconverged iterations.

increment(max_iter_so_far)

}

enqueue_fanouts(NextSub)

}

```

    } until ( cando == FALSE )
  }
}
■

```

SECTION 8.4 - PARALLEL ALGORITHM TEST RESULTS

As mentioned above, the two algorithms were implemented on a 9 processor configuration of the Sequent Balance 8000 computer (larger configurations are available). The results from several experiments for the two algorithms are given in Tables 8.1 and 8.2. As the results from the Eprom and microprocessor control circuit indicate, the timepoint pipelining algorithm makes much more efficient use of the available processors. In fact, as Table 8.2 shows, the timepoint pipelining algorithm running on the Balance 8000 runs substantially faster than the serial WR algorithm running on a Vax/780.

A second point should be made about the timepoint pipelining examples. It can be seen that the speed-up does not remain linear to nine processors, but starts to drop off. This is surprising given the size of the examples, but not when the type of circuit being simulated is considered. For the biggest example, the cmos ram, the partitioning algorithm produces approximately 75 subcircuits, and this would indicate that a speed-up of 75 should be obtainable, or at least approachable. This ignores one of the features of the WR algorithm, in that only those portions of the circuit that are active are participating in the computation. For digital circuits, this is usually less than ten percent of the circuit. This implies that for the cmos ram example over any given interval, roughly seven subcircuits are active, and involved in the computation, and therefore only a speed-up of seven could be expected.

Table 8.1 - G-S/G-J WR ON SEVERAL # OF PROCESSORS					
Circuit	FET's	1	3	6	9
uP Control	66	595	338	270	259
Eprom	348	512	317	286	266

Table 8.2 - TIMEPOINT PIPELINING WR ON SEVERAL # OF PROCESSORS						
Circuit	FET's	1	3	6	9	VAX/780
uP Control	116	704	247	159	149	240
Eprom	348	745	265	185	182	212
Cmos Ram	428	3379	1217	642	496	960

CHAPTER 9 - CONCLUSIONS

In this thesis, a wide variety of new theoretical and practical results relating to numerical integration methods for circuit simulation problems have been presented. A novel property that can be used to classify integration methods, that of *domain of dependence*, was introduced, and its importance demonstrated by example. A wide collection of integration methods that have been used for circuit simulation were then analyzed with respect to this and several other properties.

Following, the WR algorithm was introduced, and a new proof of the WR convergence, one that demonstrates that the WR algorithm is a contraction mapping in a particular norm, was presented. Extensions to the WR algorithm, along with convergence theorems, were also presented. In addition, the interaction between WR algorithms and multistep integration methods was considered in detail, and the first theorem proving the convergence of the multi-rate discretized WR relaxation algorithm was presented.

The practical aspects of WR were examined using a new circuit simulation program, RELAX2.3. The novel algorithms used by the program to partition large circuits and dynamically adjust the windows were described, and results from the program on industrial circuits presented. In addition, the implementation of two WR-based parallel circuit simulation algorithms were presented along with results.

There are several theoretical questions about WR that were only partially answered in this thesis. In particular, research is needed to more thoroughly understand the nature of WR convergence under discretization, and to characterize systems for which WR algorithms contract in uniform norm. In addition, theoretical and practical work needs to be continued on breaking large systems into smaller subsystems in such a way that relaxation algorithms converge rapidly.

There is also much work to be done to improve the speed and robustness of the WR algorithm. In particular, more sophisticated partitioning algorithms should be devised. Also, the results on parallel WR algorithms presented in this thesis are preliminary. Experiments should be carried out on a

variety of different architectures to investigate the relationships between algorithms and computer architecture.

REFERENCES

- [1] C. William Gear, *Numerical Initial Value Problems for Ordinary Differential Equations*, Prentice Hall, 1974.
- [2] L.W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Electronics Research Laboratory Rep. No. ERL-M520, University of California, Berkeley, May 1975.
- [3] W. T. Weeks, A. J. Jimenez, G. W. Mahoney, D. Mehta, H. Qassemzadeh and T. R. Scott, "Algorithms for ASTAP -- A Network Analysis Program," *IEEE Trans. on Circuit Theory*, Vol. CT-20, pp. 628-634, Nov. 1973
- [4] K. Sakallah and S.W. Director, "An Activity-Directed Circuit Simulation Algorithm," *Proc. IEEE Int. Conf. on Circ. and Computers*, October 1980, pp.1032-1035
- [5] G. De Micheli, A. Sangiovanni-Vincentelli and A.R. Newton, "New Algorithms for the Timing Analysis of Large Circuits" *Proc. 1980 Int. Symp. on Circ. and Syst.*, Houston, 1980.
- [6] G. De Micheli and A. Sangiovanni-Vincentelli "Characterization of Integration Algorithms for the Timing Analysis of MOS VLSI Circuits" *Circuit Theory and Applications*, Vol. 10, pp. 299-309, 1982
- [7] B.R. Chawla, H.K. Gummel, and P. Kozah, "MOTIS - an MOS Timing Simulator," *IEEE Trans. Circuits and Systems*, Vol. 22, pp. 901-909, 1975
- [8] C. F. Chen and P. Subramanyam, "The Second Generation MOTIS Timing Simulator -- An Efficient and Accurate Approach for General MOS Circuits" *Proc. 1984 Int. Symp. on Circ. and Syst.*, Montreal, Canada, May 1984.
- [9] Keepin, W. N., "Multirate Integration of Two Time-Scale Systems," *Ph.D. Dissertation, University of Arizona*, 1980
- [10] C. William Gear, "Automatic Multirate Methods for Ordinary Differential Equations" *Information Processing 80*, North-Holland Pub. Co., 1980

- [11] E. Lelarasmee, A. E. Ruehli, A. L. Sangiovanni-Vincentelli, "The Waveform Relaxation Method for Time Domain Analysis of Large Scale Integrated Circuits," *IEEE Trans. on CAD of IC and Syst.*, Vol. 1, n. 3, pp.131-145, July 1982.
- [12] E. Lelarasmee, "The Waveform Relaxation Method for the Time Domain Analysis of Large Scale Nonlinear Dynamical Systems", *Ph.D. dissertation, University of California, Berkeley.*
- [13] E. Lelarasmee and A. Sangiovanni-Vincentelli, "Some New Results on Waveform Relaxation Algorithms for the Simulation of Integrated Circuits," *Proc. of 1982 IEEE Int. Large Scale System Symposium*, Virginia Beach, Oct. 1982, pp. 371-376
- [14] E. Lelarasmee and A. Sangiovanni-Vincentelli, "Relax: A New Circuit Simulator for Large Scale MOS Integrated Circuits", *Proc. 19th Design Automation Conference*, Las Vegas, Nevada, pp. 682-690, June 1982.
- [15] P. Defebve, J. Beetem, W. Donath, H.Y. Hsieh, F. Odeh, A.E. Ruehli, P.K. Wolff, Sr., and J. White, "A Large-Scale Mosfet Circuit Analyzer Based on Waveform Relaxation" *International Conference on Computer Design*, Rye, New York, October 1984.
- [16] C. H. Carlin and A. Vachoux, "On Partitioning for Waveform Relaxation Time-Domain Analysis of VLSI Circuits" *Proc. 1984 Int. Symp. on Circ. and Syst.*, Montreal, Canada, May 1984.
- [17] J. White and A. Sangiovanni-Vincentelli, "Relax2: A Modified Waveform Relaxation Approach to the Simulation of MOS Digital Circuits" *Proc. 1983 Int. Symp. on Circuits and Systems*, Newport Beach, California, May 1983.
- [18] J. White and A. Sangiovanni-Vincentelli, "Relax2.1 - A Waveform Relaxation Based Circuit Simulation Program" *Proc. 1984 Int. Custom Integrated Circuits Conference* Rochester, New York, June 1984.
- [19] M. Guarini and O. A. Palusinski, "Integration of Partitioned Dynamical Systems using Waveform Relaxation and Modified Functional Linearization," *1983 Summer Computer Simulation Proceedings*, Vancouver, Canada, July 1983.

- [20] W.M.G. van Bokhoven, "An Activity Controlled Modified Waveform Relaxation Method" *1983 Conf. Proc. IEEE ISCAS*, Newport Beach, CA, May 1983.
- [21] J. M. Ortega and W.C Rheinbolt, *Iterative Solution of Nonlinear Equations in Several Variables* Academic Press, 1970.
- [22] O. Palusinski "Continuous Expansion Methods in Computer Aided Circuit Analysis" *Proc. 1984 Int. Custom Integrated Circuits Conference* Rochester, New York, June 1984
- [23] J. Kaye and A. Sangiovanni-Vincentelli, "Solution of piecewise linear ordinary differential equations using waveform relaxation and Laplace transforms", *Proc. 1982 Int. Conf. on Circ. and Comp.*, New York, Sept. 1982.
- [24] W.K. Chia, T.N. Trick, and I.N. Haij, "Stability and Convergence Properties of Relaxation Methods for Hierarchical Simulation of VLSI Circuits", *Proc. 1984 Int. Symp. on Circ. and Syst.*, Montreal, Canada, May 1984.
- [25] F. Odeh and D Zein, "A Semidirect Method for Modular Circuits" *1983 Conf. Proc. IEEE ISCAS*, May, 1983, Newport Beach, CA
- [26] G. Guardabassi "Subsystemwise Simulation of Discrete-Time Interconnected Dynamical Systems," *Electronics Research Laboratory Rep. No. ERL-M82/8, University of California, Berkeley, Feb. 1982.*
- [27] H. A. Antosiewicz, "Newton's method and boundary value problems", *Journal of Computer System Science* 2(2), 177-202 (1968)
- [28] R. S. Varga *Matrix Iterative Analysis* Prentice-Hall, 1962
- [29] F. Odeh, A. Ruehli, C. H. Carlin "Robustness Aspects of an Adaptive Waveform Relaxation Scheme" *Proc. 1983 Int. Conference on Computer Design*, Port Chester, New York, October, 1983
- [30] J. More "Nonlinear Generalization of Matrix Diagonal Dominance with Applications to Gauss-Seidel Iterations" *SIAM J. Numer. Anal.*, Vol. 9, No. 2, June 1972

- [31] J. White and A.L. Sangiovanni-Vincentelli, "Partitioning Algorithms and Parallel Implementations of Waveform Relaxation Algorithms for Circuit Simulation" *Proc. Int. Symp. on Circ. and Syst.*, Kyoto, Japan, June 1985
- [32] U. Miekkala and O. Nevanlinna "Convergence of Dynamic Iteration Methods for Initial Value Problems" *Report-MAT-A230, Institute of Mathematics*, Helsinki University of Technology, Finland, 1985
- [33] R. A. Saleh, J. E. Kleckner and A. R. Newton, "Iterated Timing Analysis and SPLICE1", *ICCAD'83 Digest*, Santa Clara, CA., 1983.
- [34] J. T. Deutsch "Algorithms and Architecture for Multiprocessor-Based Circuit Simulation", *Ph.D. Dissertation*, University of California, Berkeley, Electronics Research Laboratory, 1985
- [35] W. Rudin, *Functional Analysis*, McGraw-Hill, 1969.
- [36] C. A. Desoer and E. S. Kuh, *Basic Circuit Theory* McGraw-Hill, 1969.
- [37] R. Courant and D. Hilbert, *Partial Differential Equations*, Vol. 2 of *Methods of Mathematical Physics*, Interscience, N. Y., 1962
- [38] C. W. Ho, A. E. Ruehli, and P. A. Brennan, "The Modified Nodal Approach to Network Analysis" *Proc. IEEE Int. Symp. on Circuits and Systems*, 1974
- [39] J. K. Hale, *Ordinary Differential Equations* John Wiley and Sons, Inc., 1969
- [40] L. Chua and P. Lin, *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques*, Prentice-Hall, 1975
- [41] P. Yang, B.D. Epler, P. K. Chatterjee, "An Investigation of the charge conservation problem for MOSFET circuit simulation," *IEEE Journal of Solid-State Circuits*, Vol. SC-18, No. 1, pp. 128-138, Feb. 1983
- [42] Dahlquist, G. "A Special Stability Problem for Linear Multistep Methods," *BIT*, 3, pp. 27-43, 1963
- [43] E. M. Purcell, *Electricity and Magnetism*. McGraw-Hill, N.Y., 1965.

- [44] R. Courant and D. Hilbert, *Partial Differential Equations*, Vol. 2 of *Methods of Mathematical Physics*, Interscience, N. Y., 1962
- [45] A. R. Newton, "The Simulation of Large Scale Integrated Circuits," University of California, Berkeley, Electronics Research Laboratory, Memo. No. ERL-M78/52, July 1978.
- [46] R. K. Brayton and C. Conley, "Some Results on the Stability and Instability of the Backward Differentiation Methods with non-uniform Time steps," *Topics in Numerical Analysis*, Proc. Roy. Irish Acad. Conf., Academic Press, N. Y., 1972
- [47] A. L. Sangiovanni-Vincentelli, "On Decomposition of Large Scale Systems of Linear Algebraic Equations," *Proc. of JACC*, Denver, June 1979.
- [48] A. Vladimirescu and D. O. Pederson, "A Computer Program for the Simulation of Large Scale Integrated Circuits," *IEEE Proc. Int. Symp. on Circuits and Systems*, Chicago, 1981.
- [49] J. A. George, "On Block Elimination for Sparse Linear Systems," *SIAM J. Numerical Analysis*, Vol. 11, pp. 585-603, 1974.
- [50] W. L. Engl, R. Laur and H. K. Dirks, "MEDUSA -- A Simulator for Modular Circuits," *IEEE Trans. on Computer-Aided Design of Int. Circuits and Syst.* CAD-1,2, April 1982.
- [51] P. Yang, I. N. Hajj and T. N. Trick, "SLATE: A Circuit Simulation Program with Latency Exploitation and Node Tearing," *IEEE Proc. Int. Conf. on Circuits and Computers*, New York, Oct. 1980.
- [52] N. B. G. Rabbat, A. L. Sangiovanni-Vincentelli and H. Y. Hsieh, "A Multilevel Newton Algorithm with Macromodelling and Latency for the Analysis of Large-Scale Nonlinear Circuits in the Time Domain," *IEEE Trans. on Circuits and Systems*, Vol. CAS-26, pp. 733-741, Sep. 1979.
- [53] A.R. Newton and A. L. Sangiovanni-Vincentelli "Relaxation-Based Circuit Simulation" *IEEE Trans. on ED*, Vol. ED-30, N. 9, pp. 1184-1207, Sept. 1983
 also *SIAM Jour. on Scientific and Stat. Computing*, Vol. 4, N. 3, Sept. 1983
 also *IEEE Trans. on CAD of IC and Syst.*, July 1984.
- [54] W. Kahan, Private Notes, 1975.

- [55] T. Huang, "Generalization of the Implicit-Implicit-Explicit Method for Floating Capacitors" *Master's Thesis, University of California, Berkeley, 1983*
- [56] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and Marc Snir, "The NYU Ultracomputer -- Designing an MIMD Shared Memory Parallel Computer" *IEEE Transactions on Computers* Vol. C-32, No. 2, pp. 175-188, Feb. 1983.
- [57] J. Kleckner, "Advanced Mixed-Mode Simulation Techniques," *PhD. Thesis, University of California, Berkeley, 1984.*
- [58] C. De Boor, *A Practical Guide to Splines*, Springer-Verlag, 1978
- [59] O. Nevanlinna and F. Odeh, "Multiplier Techniques for Linear Multistep Methods," *Numer. Funct. Anal. and Optimiz.* 3(4), pp. 377-423, 1981.
- [60] J. White, A. L. Sangiovanni-Vincentelli, R. Saleh, A. R. Newton, K. Kundert, P. Moore, "RELAX2.3 User's Manual", to appear.
- [61] D. Mitra, "Chaotic, Asynchronous Relaxations for the Numerical Solution of Differential Equations by Parallel Processors," *AT&T Bell Laboratories Memorandum*, October, 1984
- [62] J. R. Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic," *Proc. 10th Annual Symp. on Computer Architecture*, pp. 124-131, June 1983.
- [63] W. Miranker, "A Survey of Parallelism in Numerical Analysis," *SIAM Review*, No. 13, pp 524-547, 1971
- [64] R. Saleh, Private Communication, 1985
- [65] D. Webber, Private Communication, 1985

